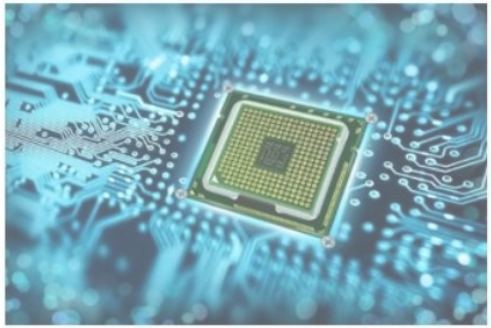
ASSEMBLEUR PIC18

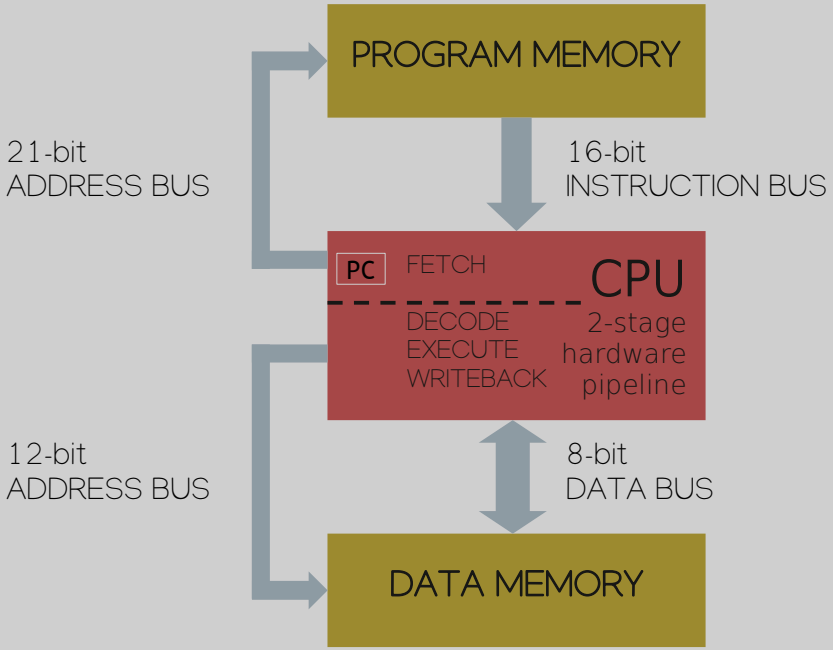



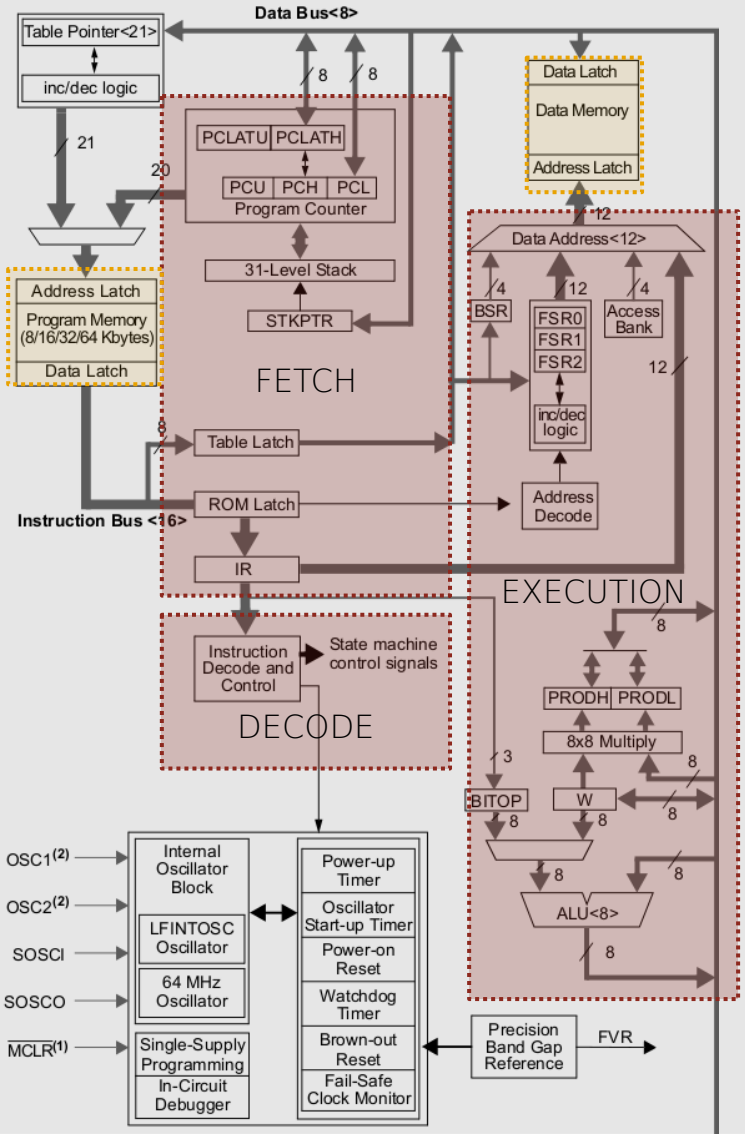





neku - hugo descoubes – enseignant Systèmes Embarqués – ENSICAEN - France
GNU\Linux Ubuntu 20.04 LTS – LibreOffice 6.4.6.2 – 2023

Architecture processeur MCU PIC18





Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word MSb	LSb	Status Affected	Notes
BYTE-ORIENTED OPERATIONS						
ADDWF f, d, a	Add WREG and f	1	0010 01da ffff ffff		C, DC, Z, OV, N	1, 2
ADDWFC f, d, a	Add WREG and Carry bit to f	1	0010 00da ffff ffff		C, DC, Z, OV, N	1, 2
ANDWF f, d, a	AND WREG with f	1	0001 01da ffff ffff		Z, N	1,2
CLRF f, a	Clear f	1	0110 101a ffff ffff		Z	2
COMF f, d, a	Complement f	1	0001 11da ffff ffff		Z, N	1, 2
CPFSEQ f, a	Compare f with WREG, skip =	1 (2 or 3)	0110 001a ffff ffff		None	4
CPFSGT f, a	Compare f with WREG, skip >	1 (2 or 3)	0110 010a ffff ffff		None	4
CPFSLT f, a	Compare f with WREG, skip <	1 (2 or 3)	0110 000a ffff ffff		None	1, 2
DECF f, d, a	Decrement f	1	0000 01da ffff ffff		C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010 11da ffff ffff		None	1, 2, 3, 4
DCFSNZ f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100 11da ffff ffff		None	1, 2
INCF f, d, a	Increment f	1	0010 10da ffff ffff		C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011 11da ffff ffff		None	4
INFSNZ f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100 10da ffff ffff		None	1, 2
IORWF f, d, a	Inclusive OR WREG with f	1	0001 00da ffff ffff		Z, N	1, 2
MOVF f, d, a	Move f	1	0101 00da ffff ffff		Z, N	1
MOVFF f_s, f_d	Move f_s (source) to 1st word f_d (destination) 2nd word	2	1100 ffff ffff ffff 1111 ffff ffff ffff		None	
MOVWF f, a	Move WREG to f	1	0110 111a ffff ffff		None	
MULWF f, a	Multiply WREG with f	1	0000 001a ffff ffff		None	1, 2
NEGF f, a	Negate f	1	0110 110a ffff ffff		C, DC, Z, OV, N	
RLCF f, d, a	Rotate Left f through Carry	1	0011 01da ffff ffff		C, Z, N	1, 2
RLNCF f, d, a	Rotate Left f (No Carry)	1	0100 01da ffff ffff		Z, N	
RRCF f, d, a	Rotate Right f through Carry	1	0011 00da ffff ffff		C, Z, N	
RRNCF f, d, a	Rotate Right f (No Carry)	1	0100 00da ffff ffff		Z, N	
SETF f, a	Set f	1	0110 100a ffff ffff		None	1, 2
SUBFWB f, d, a	Subtract f from WREG with borrow	1	0101 01da ffff ffff		C, DC, Z, OV, N	
SUBWF f, d, a	Subtract WREG from f	1	0101 11da ffff ffff		C, DC, Z, OV, N	1, 2
SUBWFB f, d, a	Subtract WREG from f with borrow	1	0101 10da ffff ffff		C, DC, Z, OV, N	
SWAPF f, d, a	Swap nibbles in f	1	0011 10da ffff ffff		None	4
TSTFSZ f, a	Test f, skip if 0	1 (2 or 3)	0110 011a ffff ffff		None	1, 2
XORWF f, d, a	Exclusive OR WREG with f	1	0001 10da ffff ffff		Z, N	

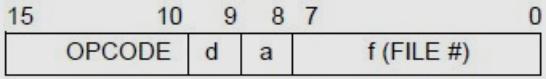
Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
BIT-ORIENTED OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	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
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	2	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call subroutine 1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	4
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software device Reset	1	0000	0000	1111	1111	All	
RETFIE	s	Return from interrupt enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	

Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word MSb LSb	Status Affected	Notes
LITERAL OPERATIONS					
ADDLW k	Add literal and WREG	1	0000 1111 kkkk kkkk	C, DC, Z, OV, N	
ANDLW k	AND literal with WREG	1	0000 1011 kkkk kkkk	Z, N	
IORLW k	Inclusive OR literal with WREG	1	0000 1001 kkkk kkkk	Z, N	
LFSR f, k	Move literal (12-bit) 2nd word to FSR(f) 1st word	2	1110 1110 00ff kkkk 1111 0000 kkkk kkkk	None	
MOVLB k	Move literal to BSR<3:0>	1	0000 0001 0000 kkkk	None	
MOVLW k	Move literal to WREG	1	0000 1110 kkkk kkkk	None	
MULLW k	Multiply literal with WREG	1	0000 1101 kkkk kkkk	None	
RETLW k	Return with literal in WREG	2	0000 1100 kkkk kkkk	None	
SUBLW k	Subtract WREG from literal	1	0000 1000 kkkk kkkk	C, DC, Z, OV, N	
XORLW k	Exclusive OR literal with WREG	1	0000 1010 kkkk kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS					
TBLRD*	Table Read	2	0000 0000 0000 1000	None	
TBLRD*+	Table Read with post-increment		0000 0000 0000 1001	None	
TBLRD*-	Table Read with post-decrement		0000 0000 0000 1010	None	
TBLRD+*	Table Read with pre-increment		0000 0000 0000 1011	None	
TBLWT*	Table Write	2	0000 0000 0000 1100	None	
TBLWT*+	Table Write with post-increment		0000 0000 0000 1101	None	
TBLWT*-	Table Write with post-decrement		0000 0000 0000 1110	None	
TBLWT+*	Table Write with pre-increment		0000 0000 0000 1111	None	

ISA de 77 instructions

Format binaire ou opcode (code opératoire) des instructions

Byte-oriented file register operations

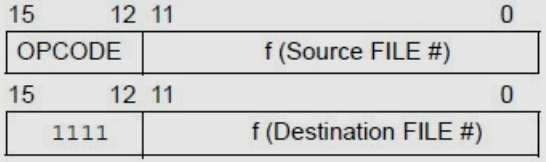


d = 0 for result destination to be WREG register
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Example Instruction

```
ADDWF MYREG, W, B
```

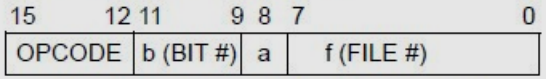
Byte to Byte move operations (2-word)



f = 12-bit file register address

```
MOVFF MYREG1, MYREG2
```

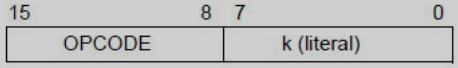
Bit-oriented file register operations



b = 3-bit position of bit in file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

```
BSF MYREG, bit, B
```

Literal operations

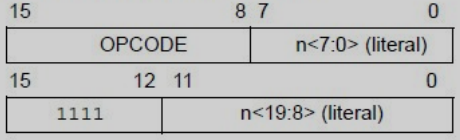


k = 8-bit immediate value

```
MOVLW 7Fh
```

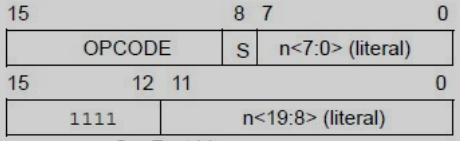
Control operations

CALL, GOTO and Branch operations



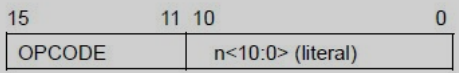
n = 20-bit immediate value

```
GOTO Label
```

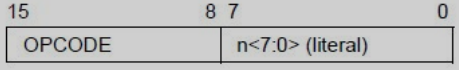


S = Fast bit

```
CALL MYFUNC
```

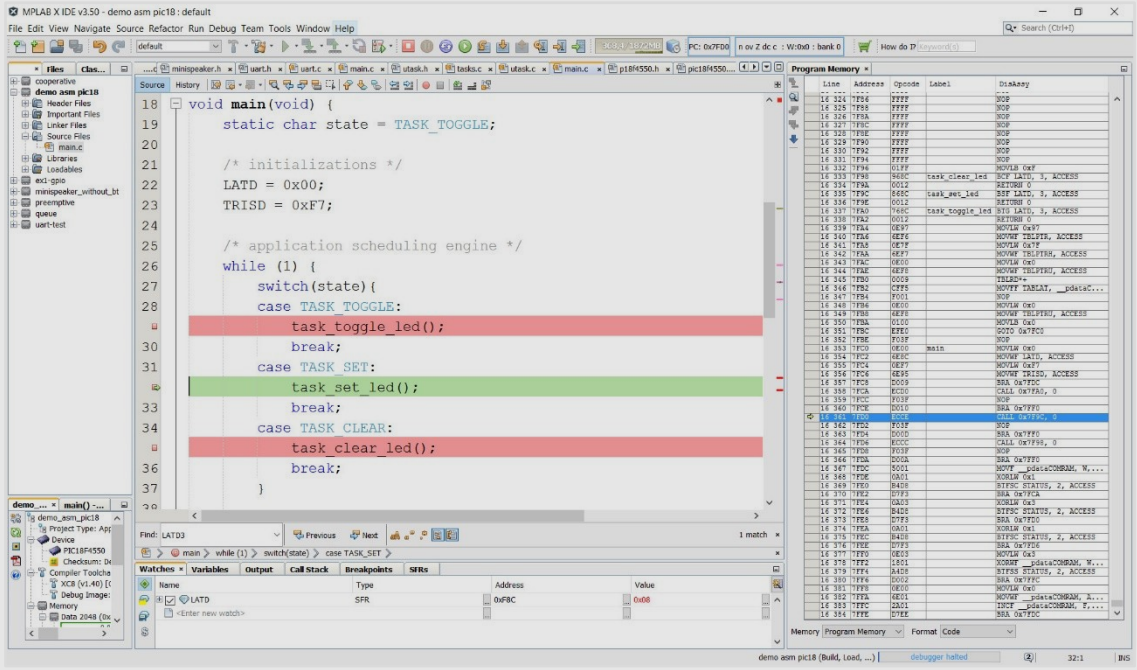



```
BRA MYFUNC
```




```
BC MYFUNC
```

Afin d'appréhender le jeu d'instruction, nous allons traduire un programme C en assembleur PIC18. Ce travail peut être répété depuis chez vous en installant l'IDE MPLABX avec les toolchains C XC8 et C18 tout en utilisant le mode simulation (cf. moodle)



```
/* CPU specific features configuration */
#pragma config FEXTOSC = OFF CLKOUTEN = OFF
#pragma config RSTOSC = HFINTOSC_64MHZ
#pragma config MCLRE = EXTMCLR PWRTE = OFF
#pragma config BOREN = SBORDIS DEBUG = OFF

#include <pic18f27K40.h>

#define TASK_TOGGLE    1
#define TASK_SET       2
#define TASK_CLEAR           3

void task_toggle_led_D2 (void);
void task_set_led_D2 (void);
void task_clear_led_D2 (void);

void main(void) {
     static char state;

     /* system init */
     state = TASK_TOGGLE;
     LATA = 0x00;
     TRISA = 0b0000000;

      /* scheduling engine */
     while (1) {

     switch(state){
          case TASK_TOGGLE:
               task_toggle_led_D2();
               break;
```

Programme à traduire


```
          case TASK_SET:
               task_set_led_D2();
               break;
          case TASK_CLEAR:
               task_clear_led_D2();

               break;
          }

     /* state machine */
     if (state == TASK_CLEAR)
          state = 0;
     state++;
     }
}

void task_toggle_led_D2 (void) {
     #asm
            BTG  LATA, 4
     #endasm
}

void task_set_led_D2 (void) {
     LATA |= 0x10;
}

void task_clear_led_D2 (void) {
     LATAbits.LATA4 = 0;
}
```

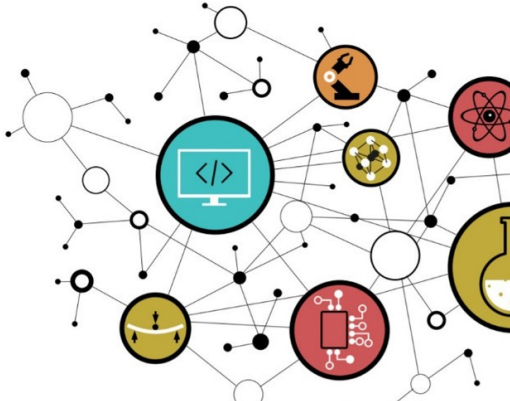
- Insertion ASM dans C
- Allocation statique de variable
- Adressage immédiat
- Instruction de contrôle
- Instructions orientées octet
- Instructions orientées bit
- Solution ASM
- Divers

De façon générale, les squelettes des applications seront développés en C. Ceci facilite la lisibilité, l'édition et la maintenance du code. L'assembleur sera en générale utilisé afin de développer des procédures ou fonctions spécifiques, le plus souvent dans une optique d'optimisation (empreinte mémoire du code et/ou accélération d'un traitement)

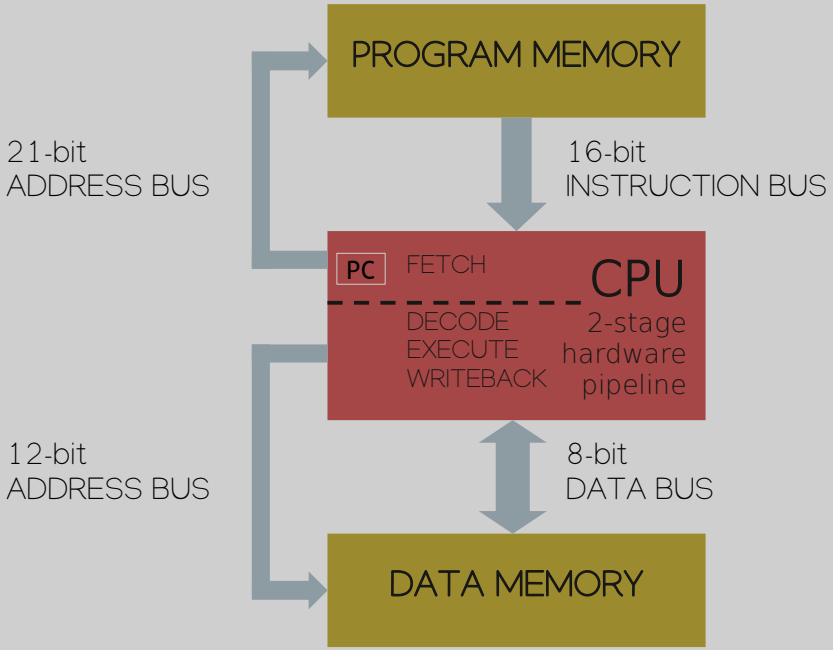
```
void task_toggle_led_D2 (void) {
     static char toggle = 0;

     if ( toggle != 0) {
          toggle = 0;
          LATAbits.LATA4 = 0;
     } else {
          toggle = 1;
          LATAbits.LATA4 = 1;
     }
}
```

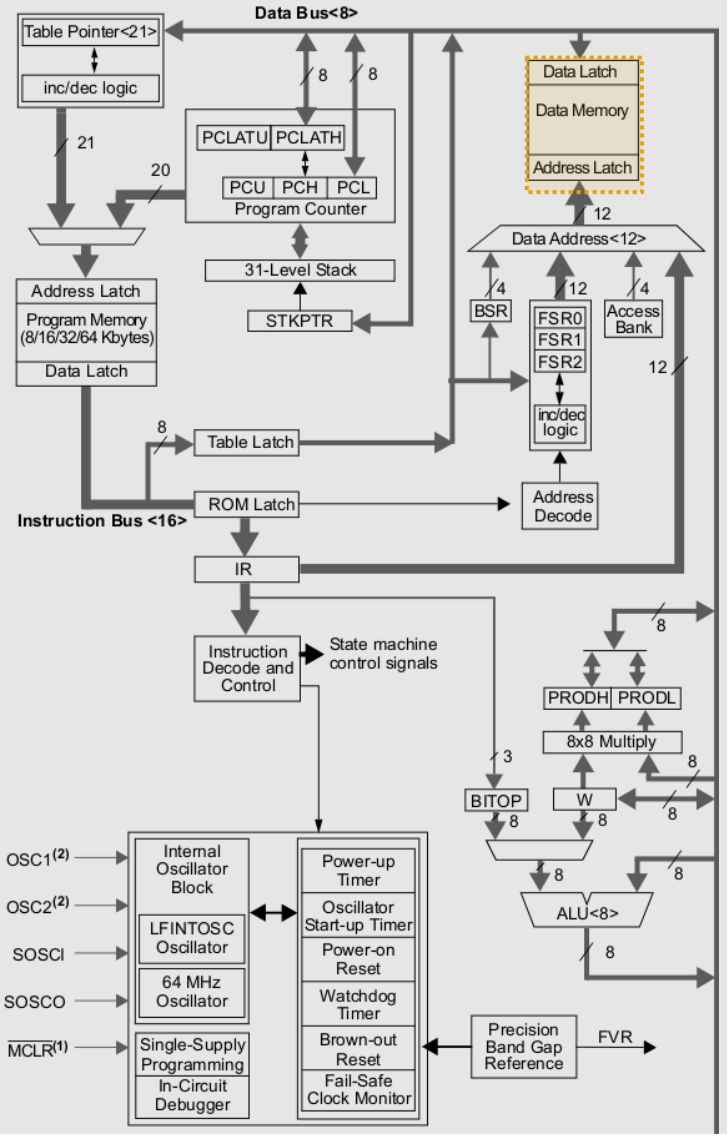
```
void task_toggle_led_D2 (void) {
    #asm
        BTG  LATA, 4
    #endasm
}
```

- Insertion ASM dans C
- **Allocation statique de variable**
- Adressage immédiat
- Instruction de contrôle
- Instructions orientées octet
- Instructions orientées bit
- Solution ASM
- Divers



Architecture processeur MCU PIC18





Afin de faciliter la compréhension du jeu d'instruction, toutes les allocations en assembleur de variables en mémoire donnée seront statiques (dans cet exercice). Adresse générée à l'édition des liens et connue pendant la totalité de la durée de vie du programme. *Nous ne regarderons pas l'utilisation de la pile (stack) et n'allouerons pas de variables locales hors static (vu en cours d'architectures des ordi.).*

En partant d'un programme assembleur PIC18, chez Microchip il existe 2 solutions technologiques pour réaliser une conversion binaire. Par la toolchain C XC8 ou le programme assembleur MPASMWIN

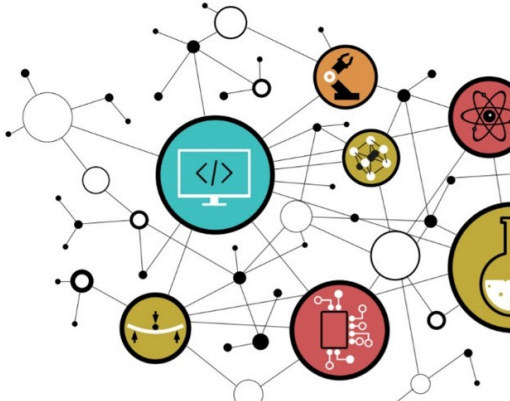
XC8

```
; reserve 1 byte in access bank
PSECT <static_section_name>,class=BANK0,space=1
state:  ds  1
```

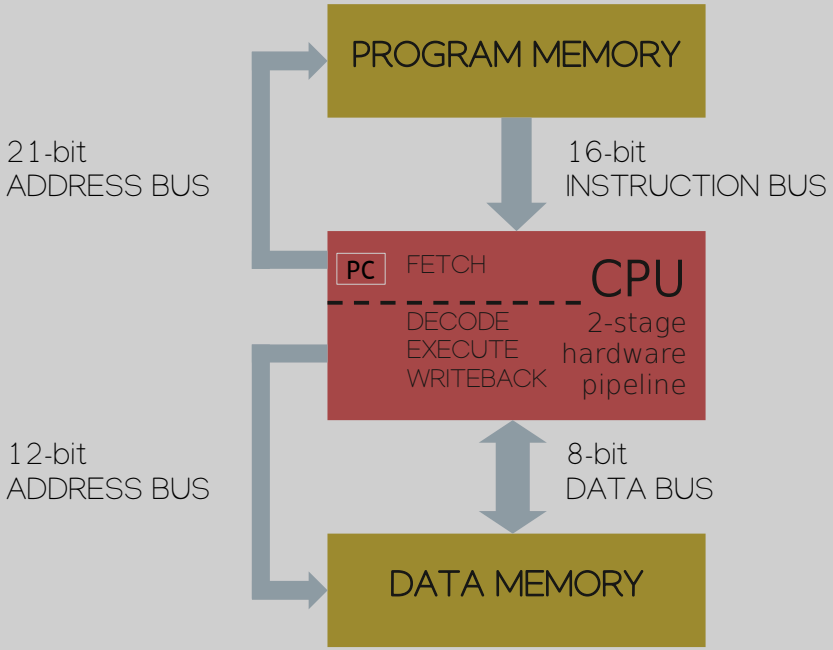
MPASMWIN

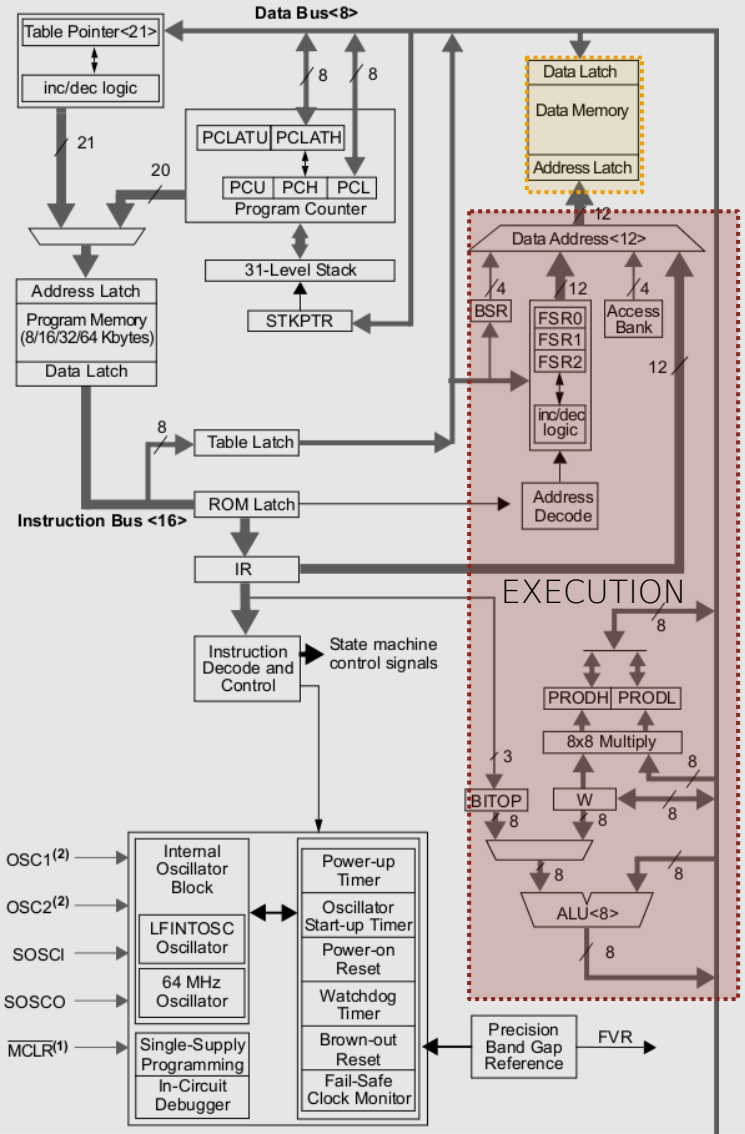
```
; reserve 1 byte in access bank
    idata_acs
state   db      0
```

- Insertion ASM dans C
- Allocation statique de variable
- **Adressage immédiat**
- Instruction de contrôle
- Instructions orientées octet
- Instructions orientées bit
- Solution ASM
- Divers



Architecture processeur MCU PIC18





Une instruction utilisant un adressage immédiat manipule directement une constante. Le binaire correspondant à la constante est encapsulé dans le code binaire de l'instruction. Sur architecture RISC 32bits, il est souvent limité à des constantes sur 16bits. L'adressage immédiat est nommé *literal operation* sur architecture PIC18

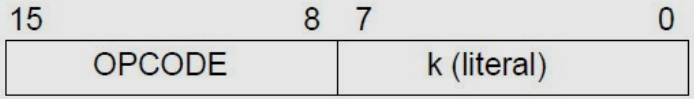
PIC18 C Program	PIC18 assembler Program

`/* system init */`
`state = TASK_TOGGLE`
`LATA = 0x00;`
`TRISA = 0b00000000;`
	`MOVLW 1 ; adressage immédiat`
	`MOVWF state ; adressage direct`
	`MOVLW 0x00`
	`MOVWF LATA`
	`MOVLW 0b00000000`
	`MOVWF TRISA`

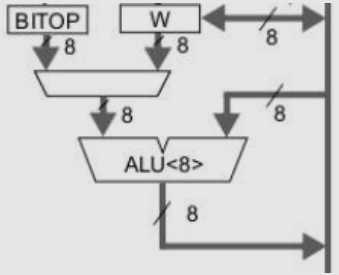
Adressage immédiat – toutes les instructions !

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb LSb	Status Affected	Notes
LITERAL OPERATIONS						
ADDLW	k	Add literal and WREG	1	0000 1111 kkkk kkkk	C, DC, Z, OV, N	
ANDLW	k	AND literal with WREG	1	0000 1011 kkkk kkkk	Z, N	
IORLW	k	Inclusive OR literal with WREG	1	0000 1001 kkkk kkkk	Z, N	
LFSR	f, k	Move literal (12-bit) 2nd word to FSR(f) 1st word	2	1110 1110 00ff kkkk 1111 0000 kkkk kkkk	None	
MOVLB	k	Move literal to BSR<3:0>	1	0000 0001 0000 kkkk	None	
MOVLW	k	Move literal to WREG	1	0000 1110 kkkk kkkk	None	
MULLW	k	Multiply literal with WREG	1	0000 1101 kkkk kkkk	None	
RETLW	k	Return with literal in WREG	2	0000 1100 kkkk kkkk	None	
SUBLW	k	Subtract WREG from literal	1	0000 1000 kkkk kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR literal with WREG	1	0000 1010 kkkk kkkk	Z, N	

Literal operations



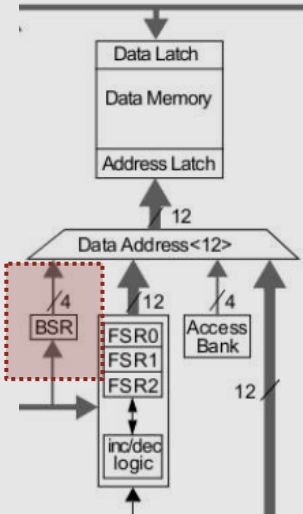
k = 8-bit immediate value



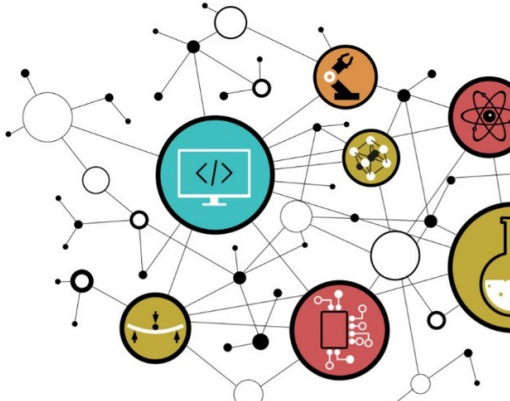
Adressage immédiat – sélection d'une banque en mémoire

Rappelons que la mémoire donnée est découpée en 16 banques de 256o. Afin d'adresser la mémoire, le CPU examine le champ <a> des instructions à adressage direct manipulant une adresse relative sur 8bits. L'exemple ci-dessous présente deux solutions afin d'adresser la banque 0. L'instruction MOVLB (MOV 4bits constant to BSR, Bank Select Register) permet d'adresser la totalité du mapping mémoire. BSR fixe les 4 bits de poids forts d'une adresse 12bits en mémoire donnée

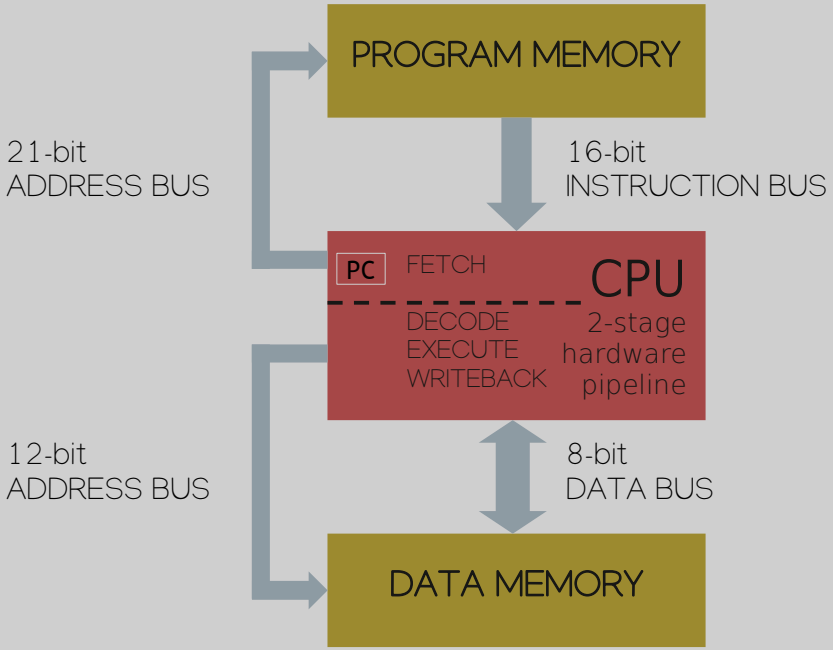
```
; access bank (bank0 or bank15)
; bank0   : 0x000-0x05F (GPR)
; bank15 : 0xF60-0xFFF  (SFR)
ADDWF       <8bits_relative_address_data_memory>, 0, 0

Autre solution :

; only bank0 is selected
MOVLB       0x0
ADDWF       <8bits_relative_address_data_memory >, 0, 1
```

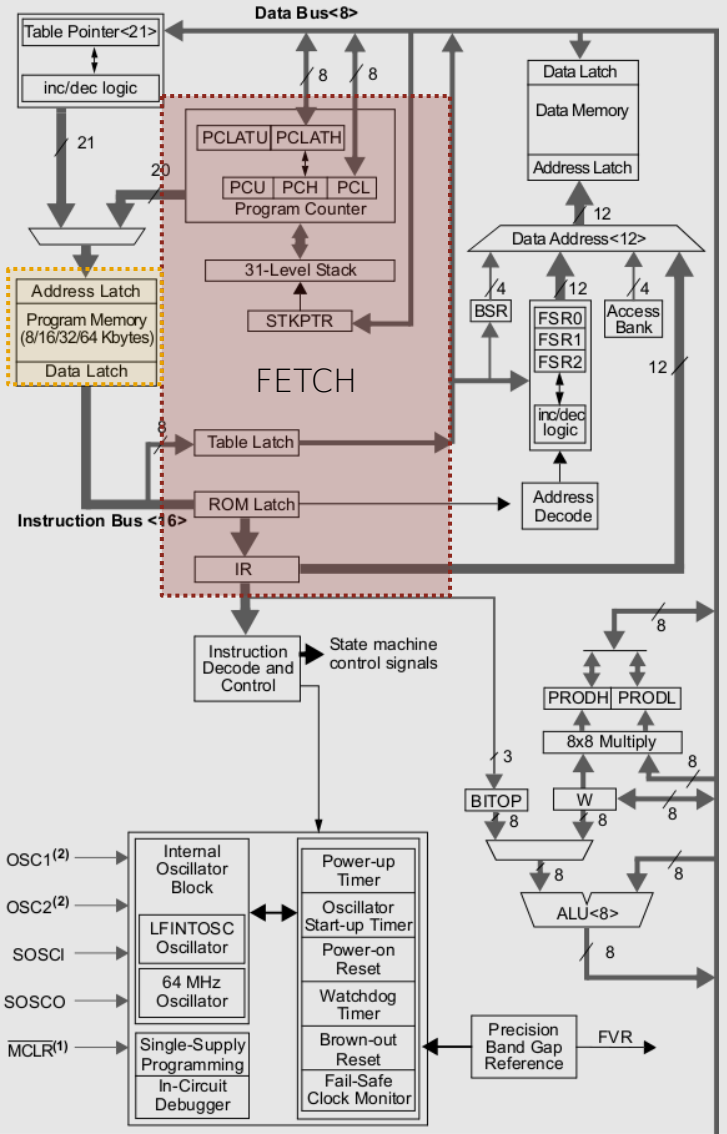


- Insertion ASM dans C
- Allocation statique de variable
- Adressage immédiat
- **Instruction de contrôle**
- Instructions orientées octet
- Instructions orientées bit
- Solution ASM
- Divers



Architecture processeur MCU PIC18





Intéressons-nous aux instructions de contrôle (if, else if, else, switch/case, while, for ...) et appels de fonction en langage C. Toutes ces instructions ont un point commun, elles réalisent un saut dans le code. Toutes ces instructions modifient le pointeur programme PC (Program Counter) présent dans l'étage de FETCH du CPU.

PIC18 C Program

```
void main(void) {
     /* user code 1 */

    while (1) {
        /* user code 2 */
     }
}
```

PIC18 assembler Program

```
main:
     ; user code 1
main_l1:
     ;user code 2
     GOTO     main_l1
```

label = référence symbolique à une adresse physique en mémoire programme résolue à l'édition des liens

La mémoire programme est adressable par octet sur 21bits, soit un espace mémoire accessible de 2Mo (ou 1Mword). 1 mot ou word pour notre processeur correspond à la taille par défaut de l'opcode la majorité des instructions, soit 2o. A titre indicatif, l'opcode de l'instruction GOTO fait 32bits ou 4o car elle implémente une opérande représentant une adresse absolue en mémoire programme sur 20bits

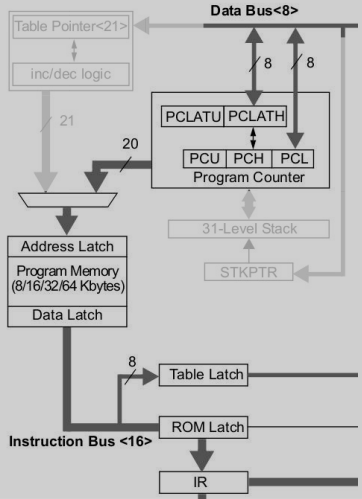
GOTO instruction Datasheet

GOTO — **Unconditional Branch**

Syntax:	[*label*] GOTO k
Operands:	$0 \le k \le 1048575$
Operation:	$k \rightarrow PC<20:1>$ $0 \rightarrow PC<0>$,
Status Affected:	None
Encoding:	

1st word (k<7:0>)	1110	1111	k_7kkk	$kkkk_0$
2nd word (k<19:8>)	1111	$k_{19}kkk$	kkkk	$kkkk_8$

PIC18F27K40 MCU architecture



L'instruction GOTO implémente un adressage absolu. L'opérande représentant l'adresse de saut fait donc 20bits (opcode sur 4o). L'instruction BRA implémente un adressage relatif à PC entre -1024o et +1023o. L'opérande représentant l'offset est codée sur 11 bits (opcode sur 2o). BRA possèdent une empreinte mémoire plus faible mais n'est pas plus rapide à l'exécution (2 cycles CPU car le pipeline du CPU doit être vidé – pipeline flush). Néanmoins, BRA ne permet pas d'adresser toute la mémoire, soit les 4Ko potentiel d'un PIC18. Elle n'implémente qu'un saut relatif par rapport à l'adresse courante (PC) durant l'exécution du BRA.

Adressage absolu

```
main:
     ; user code 1
main_l1:
     ;user code 2
     GOTO     main_l1
```

Adressage relatif

```
main:
     ; user code 1
main_l1:
     ;user code 2
     BRA      main_l1
```

Un saut conditionnel est lié au résultat d'une opération traitée précédemment par l'ALU. A chaque opération, l'ALU sauvegarde dans le registre STATUS des informations sur le résultat de l'opération (flags ou drapeaux C, DC, Z, OV et N). C ou carry précise un éventuel débordement. Z si le résultat est nul. N si le résultat est négatif. Les sauts conditionnels se font sur activation de ces flags et donc après l'utilisation d'une instruction affectant les flags

C program

```
/* application state update */
if (state == TASK_CLEAR)
     state = 0;
state++;
```

PIC18 assembler program

```
        ; application state update
        MOLLW    3
        SUBWF    state,w
        BNZ      main_l2
        MOVLW    0x00
        MOVWF    state
main_l2: INCF    state
```

Status Register

Bit	7	6	5	4	3	2	1	0
		$\overline{TO}$	$\overline{PD}$	N	OV	Z	DC	C
Access		R	R	R/W	R/W	R/W	R/W	R/W
Reset		1	1	0	0	0	0	0

Bit 6 – $\overline{TO}$ Time-Out bit
Reset States: POR/BOR = 1
All Other Resets = q

Value	Description
1	Set at power-up or by execution of `CLRWDT` or `SLEEP` instruction
0	A WDT time-out occurred

Bit 5 – $\overline{PD}$ Power-Down bit
Reset States: POR/BOR = 1
All Other Resets = q

Value	Description
1	Set at power-up or by execution of `CLRWDT` instruction
0	Cleared by execution of the `SLEEP` instruction

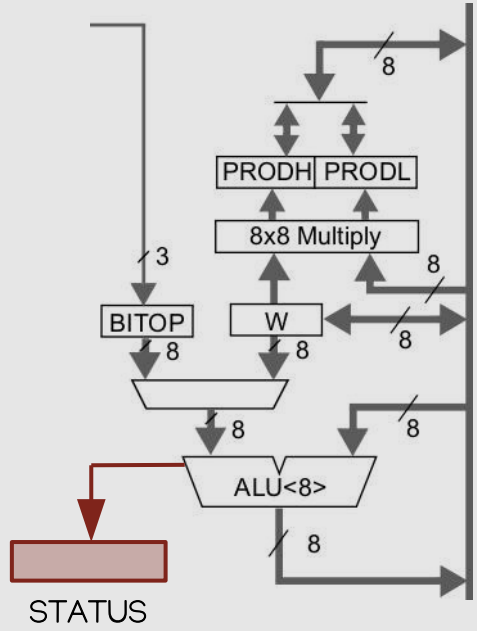
Bit 4 – N Negative bit
Used for signed arithmetic (2's complement); indicates if the result is negative, (ALU MSb = `1`).
Reset States: POR/BOR = 0
All Other Resets = u

Value	Description
1	The result is negative
0	The result is positive

Bit 3 – OV Overflow bit
Used for signed arithmetic (2's complement); indicates an overflow of the 7-bit magnitude, which causes the sign bit (bit 7) to change state.
Reset States: POR/BOR = 0
All Other Resets = u

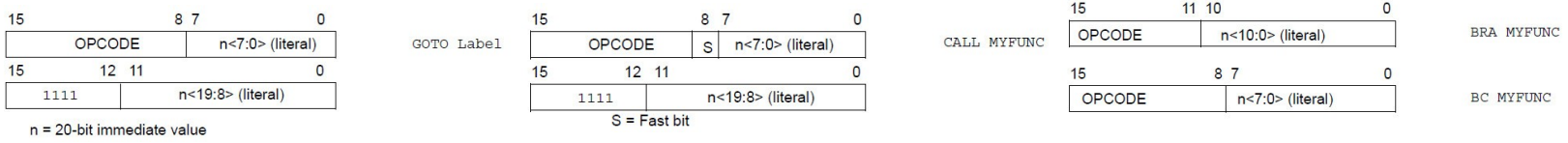
Value	Description
1	Overflow occurred for current signed arithmetic operation
0	No overflow occurred

Bit 2 – Z Zero bit
Reset States: POR/BOR = 0



Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word MSb	LSb	Status Affected
BYTE-ORIENTED OPERATIONS					
ADDWF f, d, a	Add WREG and f	1	0010 01da	ffff ffff	C, DC, Z, OV, N
ADDWFC f, d, a	Add WREG and Carry bit to f	1	0010 00da	ffff ffff	C, DC, Z, OV, N

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	2	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call subroutine 1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	4
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software device Reset	1	0000	0000	1111	1111	All	
RETFIE	s	Return from interrupt enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	



```
switch(state){
case TASK_TOGGLE:
    task_toggle_led_D2();
    break;
case TASK_SET:
    task_set_led_D2();
    break;
case TASK_CLEAR:
    task_clear_led_D2();
    break;
}
```

Fo qu'j'fé quoi !?

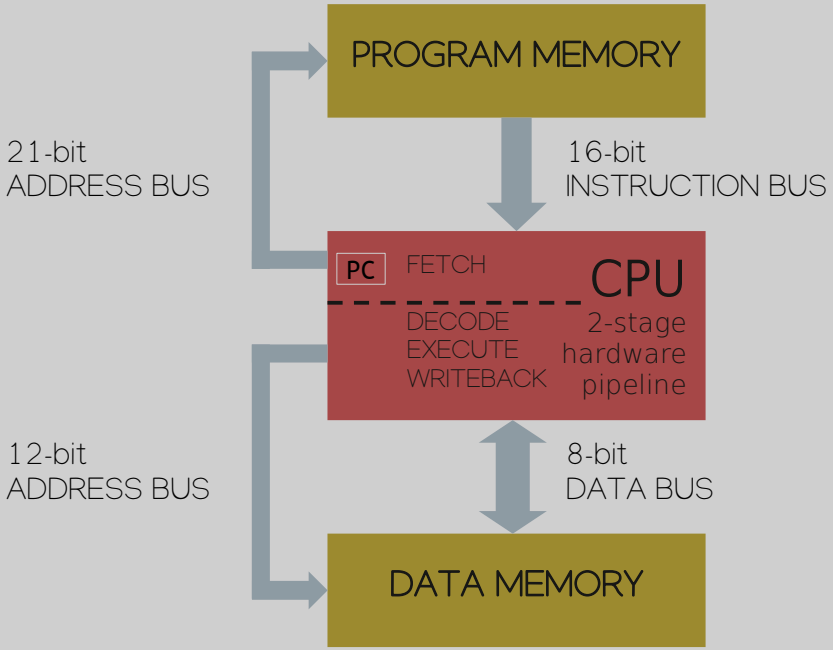
```
switch(state){
case TASK_TOGGLE:
    task_toggle_led_D2();
    break;
case TASK_SET:
    task_set_led_D2();
    break;
case TASK_CLEAR:
    task_clear_led_D2();
    break;
}
```

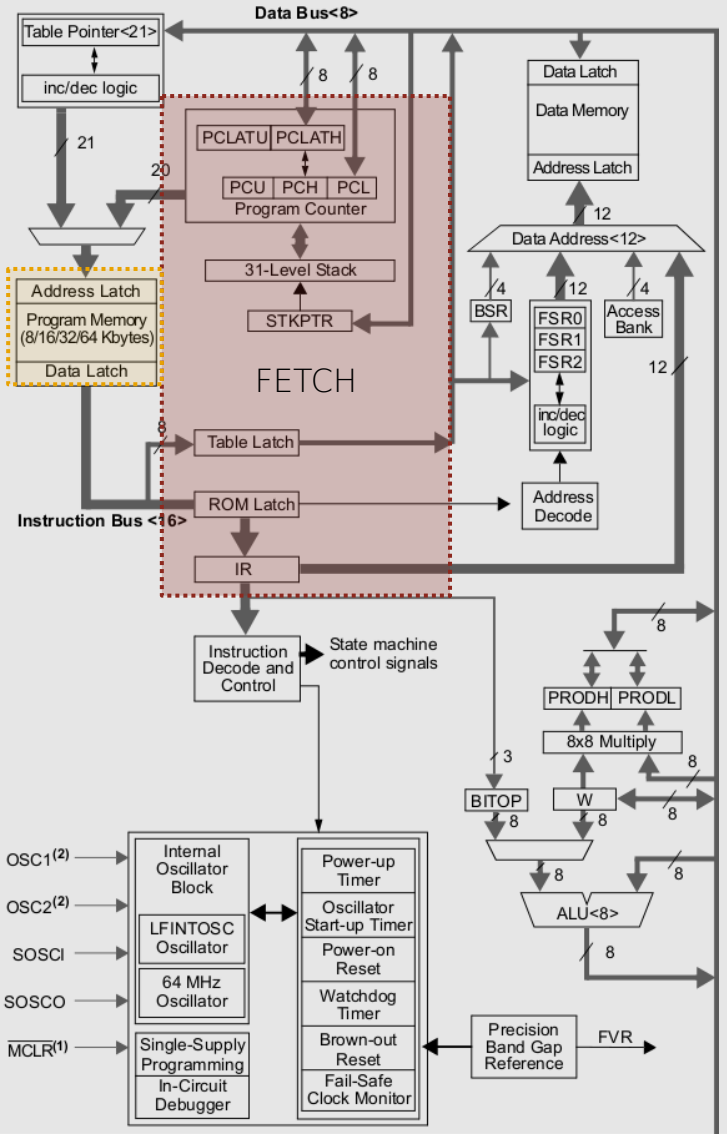
Mnemonic, Operands		Description
CONTROL OPERATIONS		
BC	n	Branch if Carry
BN	n	Branch if Negative
BNC	n	Branch if Not Carry
BNN	n	Branch if Not Negative
BNOV	n	Branch if Not Overflow
BNZ	n	Branch if Not Zero
BOV	n	Branch if Overflow
BRA	n	Branch Unconditionally
BZ	n	Branch if Zero
CALL	n, s	Call subroutine 1st word 2nd word
CLRWDT	—	Clear Watchdog Timer
DAW	—	Decimal Adjust WREG
GOTO	n	Go to address 1st word 2nd word
NOP	—	No Operation
NOP	—	No Operation
POP	—	Pop top of return stack (TOS)
PUSH	—	Push top of return stack (TOS)
RCALL	n	Relative Call
RESET		Software device Reset
RETFIE	s	Return from interrupt enable
RETLW	k	Return with literal in WREG
RETURN	s	Return from Subroutine
SLEEP	—	Go into Standby mode

Instructions de contrôle – Une solution possible !

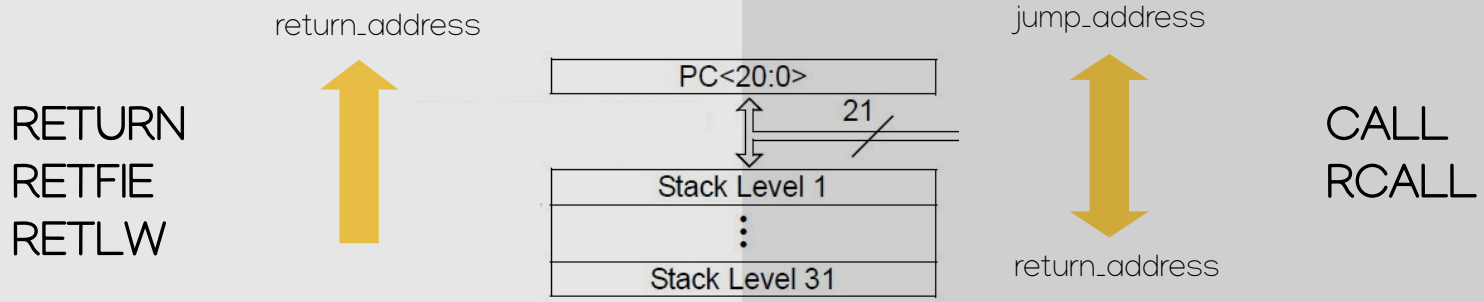
```
            MOVF        state, w
            XORLW       1
            BZ          main_c1
            MOVF        state, w
            XORLW       2
            BZ          main_c2
            MOVF        state, w
            XORLW       3
            BZ          main_c3
main_c1:    CALL        task_toggle_led_D2
            BRA         main_e1
main_c2:    CALL        task_set_led_D2
            BRA         main_e1
main_c3:    CALL        task_clear_led_D2
main_e1:    …
```

Architecture processeur MCU PIC18





L'instruction CALL implémente un adressage absolu (opcode 4o), contrairement à RCALL utilisant un adressage relatif à PC avec un offset de -1024/+1023o (opcode 2o). Les instructions CALL et RCALL modifient PC mais réalisent également une écriture sur la pile matérielle de 31 niveaux. L'adresse de retour est sauvée (PC+4 pour CALL et PC+2 pour RCALL). Par exemple, l'appel de l'instruction RETURN dépile l'adresse de retour pour l'écrire dans PC



CALL **Call Subroutine**

Syntax: [*label*] CALL k, s

Operands: $0 \le k \le 1048575$
$s \in [0,1]$

Operation: (PC)+ 4 → TOS,
k → PC<20:1>,
0 → PC<0>,
if s = 1
(WREG) → WREGS,
(STATUS) → STATUSS,
(BSR) → BSRS

Status Affected: None

Encoding:

1st word (k<7:0>)	1110	110s	$k_7$kkk	kkkk_0
2nd word (k<19:8>)	1111	k_{19}kkk	kkkk	kkkk_8

Description: Subroutine call of entire 2M byte memory range. First, return address (PC+ 4) is pushed onto the return stack (20-bits wide).

If 's' = 1, the WREG, STATUS and BSR Registers are also pushed into their respective Shadow Registers, WREGS, STATUSS and BSRS.

If 's' = 0, no update occurs.

Then the 20-bit value 'k' is loaded into PC<20:1>. CALL is a two-cycle instruction.

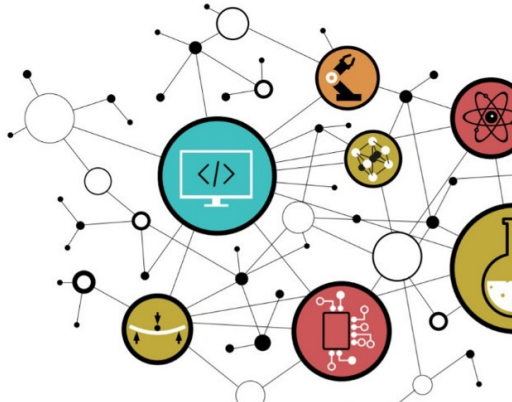
Words: 2

Cycles: 2

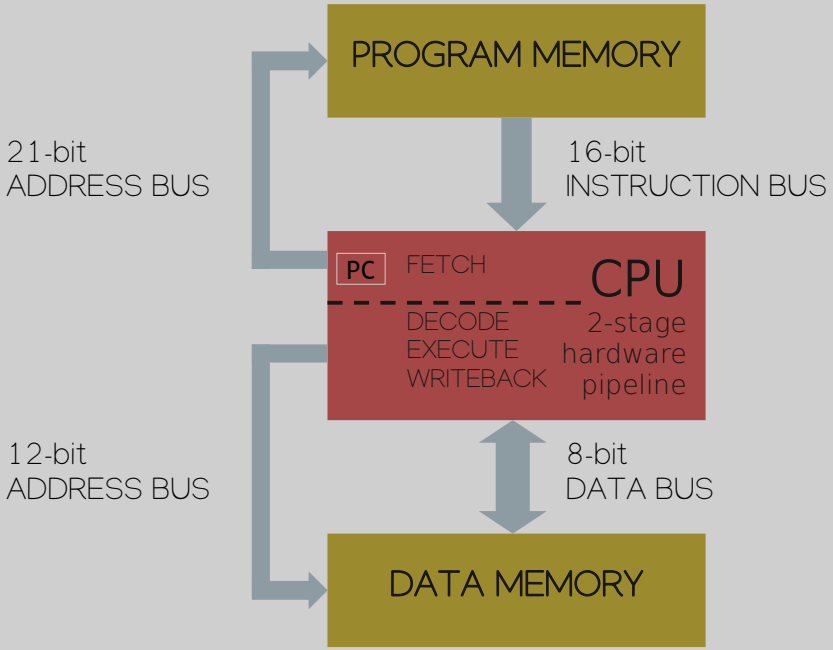
Top Of Stack pour l'adresse de retour

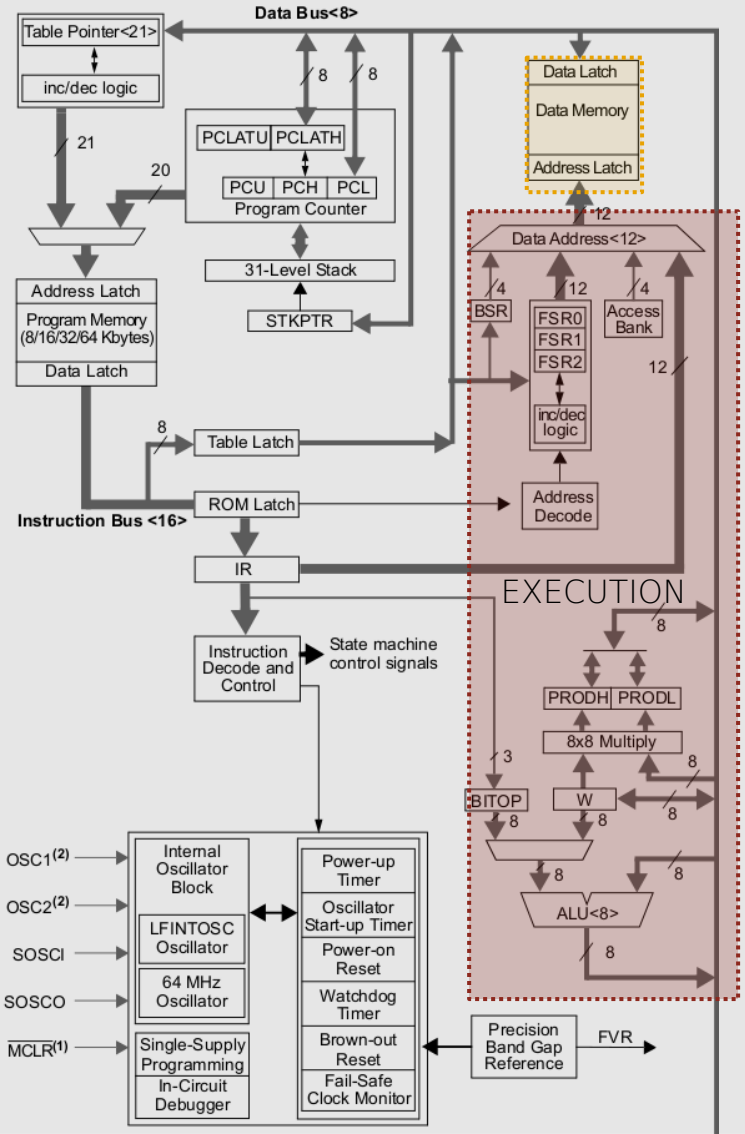
Le registres dont le nom est suffixé par S (Shadows Registers) sont cachés des développeurs et utilisés durant les sauvegardes de contexte matérielles avec l'usage d'interruption. Cette méthode accélère le mécanise de commutation

- Insertion ASM dans C
- Allocation statique de variable
- Adressage immédiat
- Instruction de contrôle
- **Instructions orientées octet**
- Instructions orientées bit
- Solution ASM
- Divers



Architecture processeur MCU PIC18





Microchip définit une famille d'instructions dites orientées octet. Cette famille regroupe les instructions arithmétiques et logiques traitées par l'ALU 8bits ou le multiplieur 8bits ainsi que les instructions de déplacement de données.

Les instructions de déplacement d'information utilisent un mode d'adressage direct afin d'effectuer des chargements et sauvegardes de données du CPU vers mémoire (MOVWF, 1cy), de la mémoire vers la mémoire (MOVFF, 2cy) ou de la mémoire vers le CPU (MOVF, 1cy).

C program

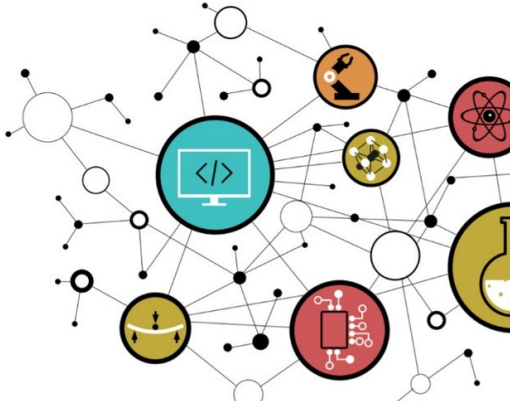
```
/* activate LED state */
void task_set_led_D2 (void) {
    LATA |= 0x10;
}
```

PIC18 assembler program

```
; activate LED state
task_set_led_D2: MOLLW    0x10
                 IORWF    LATA, f
                 RETURN
```

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb LSb	Status Affected	Notes
BYTE-ORIENTED OPERATIONS						
ADDWF	f, d, a	Add WREG and f	1	0010 01da ffff ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010 00da ffff ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001 01da ffff ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110 101a ffff ffff	Z	2
COMF	f, d, a	Complement f	1	0001 11da ffff ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110 001a ffff ffff	None	4
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110 010a ffff ffff	None	4
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110 000a ffff ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000 01da ffff ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010 11da ffff ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100 11da ffff ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010 10da ffff ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011 11da ffff ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100 10da ffff ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001 00da ffff ffff	Z, N	1, 2
MOVF	f, d, a	Move f	1	0101 00da ffff ffff	Z, N	1
MOVFF	f_s, f_d	Move f_s (source) to 1st word f_d (destination) 2nd word	2	1100 ffff ffff ffff 1111 ffff ffff ffff	None	
MOVWF	f, a	Move WREG to f	1	0110 111a ffff ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000 001a ffff ffff	None	1, 2
NEGF	f, a	Negate f	1	0110 110a ffff ffff	C, DC, Z, OV, N	
RLCF	f, d, a	Rotate Left f through Carry	1	0011 01da ffff ffff	C, Z, N	1, 2
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100 01da ffff ffff	Z, N	
RRCF	f, d, a	Rotate Right f through Carry	1	0011 00da ffff ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100 00da ffff ffff	Z, N	
SETF	f, a	Set f	1	0110 100a ffff ffff	None	1, 2
SUBFWB	f, d, a	Subtract f from WREG with borrow	1	0101 01da ffff ffff	C, DC, Z, OV, N	
SUBWF	f, d, a	Subtract WREG from f	1	0101 11da ffff ffff	C, DC, Z, OV, N	1, 2
SUBWFB	f, d, a	Subtract WREG from f with borrow	1	0101 10da ffff ffff	C, DC, Z, OV, N	
SWAPF	f, d, a	Swap nibbles in f	1	0011 10da ffff ffff	None	4
TSTFSZ	f, a	Test f, skip if 0	1 (2 or 3)	0110 011a ffff ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001 10da ffff ffff	Z, N	

- Insertion ASM dans C
- Allocation statique de variable
- Adressage immédiat
- Instruction de contrôle
- Instructions orientées octet
- **Instructions orientées bit**
- Solution ASM
- Divers



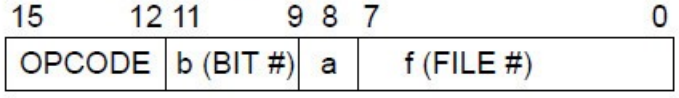
Les instructions orientées bit permettent de modifier voire tester la valeur d'un bit dans une case mémoire. Ces instructions sont très pratiques pour la configuration et la gestion de périphériques même si elles restent peu rencontrées sur grand nombre d'architectures actuelles. Nous devons spécifier l'adresse de la case mémoire (adresse relative à une banque sur 8bits) et la position du bit dans l'octet (entre 0 et 7)

C program	`/* toggle LED state */`
	`void task_toggle_led_D2 (void) {`
	`  #asm`
	`  BTG  LATA, 4`
	`  #endasm`
	`}`
		`/* inactivate LED state */`
		`void task_clear_led_D2 (void) {`
		`  LATAbits.LATA4 = 0;`
		`}`

PIC18 assembler program	`task_toggle_led_D2: BTG  LATA, 4` `RETURN`	`task_clear_led_D2: BCF  LATA, 4` `RETURN`

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb LSb	Status Affected	Notes
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

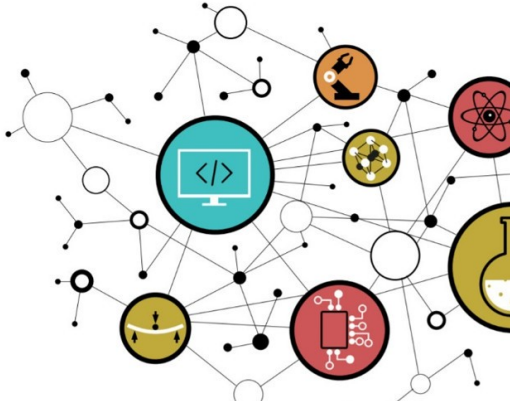
Bit-oriented file register operations



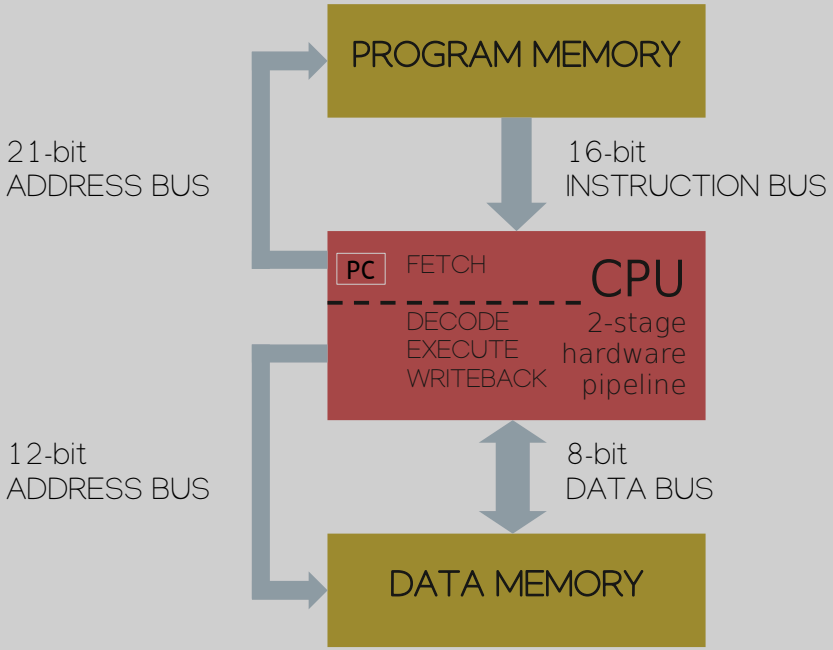

```
BSF MYREG, bit, B
```

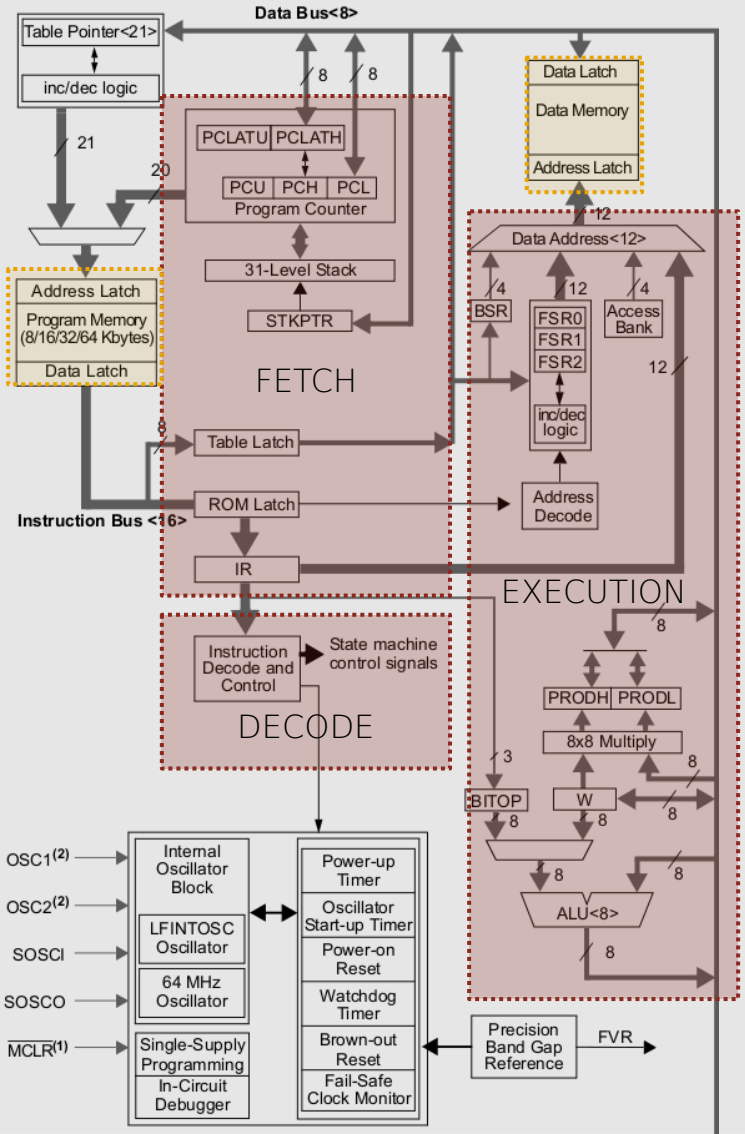
b = 3-bit position of bit in file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

- Insertion ASM dans C
- Allocation statique de variable
- Adressage immédiat
- Instruction de contrôle
- Instructions orientées octet
- Instructions orientées bit
- **Solution ASM**
- Divers



Architecture processeur MCU PIC18





```
/* CPU specific features configuration */
#pragma config FEXTOSC = OFF CLKOUTEN = OFF
#pragma config RSTOSC = HFINTOSC_64MHZ
#pragma config MCLRE = EXTMCLR PWRTE = OFF
#pragma config BOREN = SBORDIS DEBUG = OFF

#include <pic18f27K40.h>

#define TASK_TOGGLE    1
#define TASK_SET       2
#define TASK_CLEAR           3

void task_toggle_led_D2 (void);
void task_set_led_D2 (void);
void task_clear_led_D2 (void);

void main(void) {
     static char state;

     /* system init */
     state = TASK_TOGGLE;
     LATA = 0x00;
     TRISA = 0b0000000;

      /* scheduling engine */
     while (1) {

     switch(state){
          case TASK_TOGGLE:
               task_toggle_led_D2();
               break;
```

Programme à traduire


```
          case TASK_SET:
               task_set_led_D2();
               break;
          case TASK_CLEAR:
               task_clear_led_D2();

               break;
          }

     /* state machine */
     if (state == TASK_CLEAR)
          state = 0;
     state++;
     }
}

void task_toggle_led_D2 (void) {
     #asm
            BTG  LATA, 4
     #endasm
}

void task_set_led_D2 (void) {
     LATA |= 0x10;
}

void task_clear_led_D2 (void) {
     LATAbits.LATA4 = 0;
}
```

```
; CPU features
CONFIG FEXTOSC = OFF RSTOSC = HFINTOSC_64MHZ
CONFIG MCLRE = EXTMCLR DEBUG = OFF

#include <p18f27k40.inc>

; private declaration
    idata_acs
state       db    0

TASK_TOGGLE       equ   1
TASK_SET          equ   2
TASK_CLEAR        equ   3

; reset interrupt vector
     org     0x000000
reset_v:    GOTO        main


; application entry point
main:       MOVLW       TASK_TOGGLE
            MOVWF       state
            MOVLW       0b00000000
            MOVWF       LATA
            MOVLW       0x00
            MOVWF       TRISA
main_l1:    MOVF        state,w
            XORLW       TASK_TOGGLE
            BZ          main_c1
            MOVF        state,w
            XORLW       TASK_SET
            BZ          main_c2
```

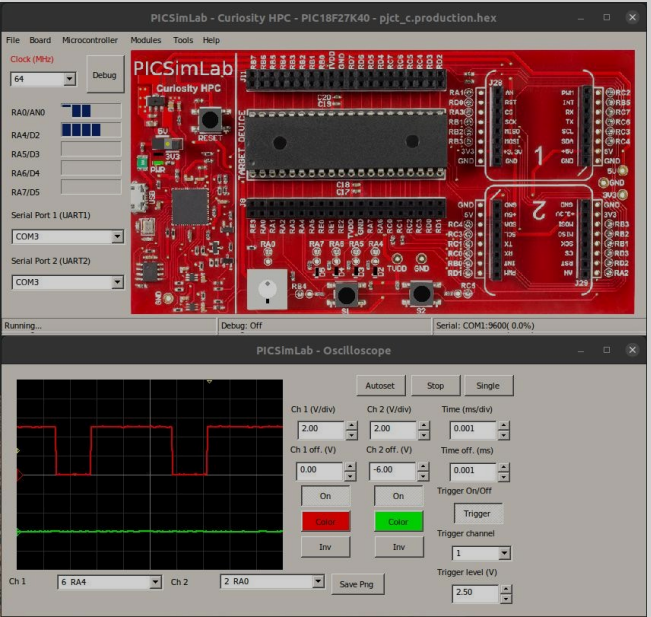


```
            MOVF        state,w
            XORLW       TASK_CLEAR
            BZ          main_c3
main_c1:    CALL        task_toggle_led_D2
            BRA         main_e1
main_c2:    CALL        task_set_led_D2
            BRA         main_e1
main_c3:    CALL        task_clear_led_D2
main_e1:    MOVLW       TASK_CLEAR
            SUBWF       state,w
            BNZ         main_l2
            MOVLW       0x00
            MOVWF       state
main_l2:    INCF        state
            BRA         main_l1

; toggle LED state
task_toggle_led_D2:
            BTG         LATA, LATA4
            RETURN

; activate LED state
task_set_led_D2:
            MOVLW       0x10
            IORWF       LATA,f
            RETURN

; inactivate LED sate
task_clear_led_D2:
            BCF         LATA, LATA4
            RETURN
            END
```

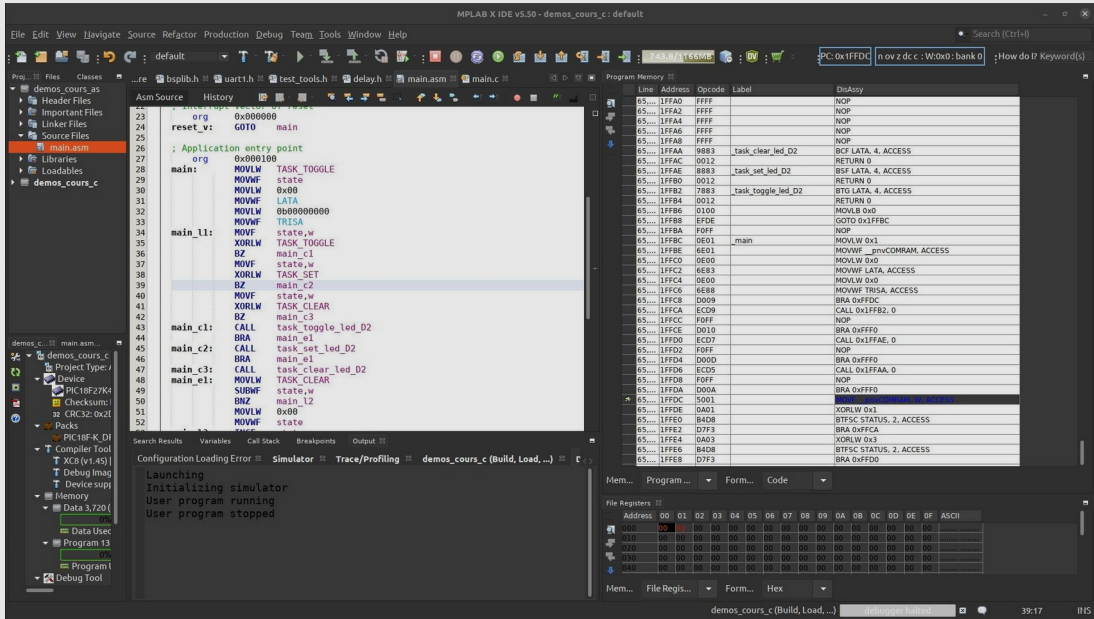
SYSTÈMES EMBARQUES
Solution complète en ASM - Simulateur

Dans le répertoire *tp/disco/apps/demos_cours*, il vous est proposé des projets pré-créés sous XC8 v1.45 (programme C et ASM PIC18). Ces projets vous permettront de pouvoir retravailler et durcir votre compréhension des processeurs PIC18 et de l'enseignement

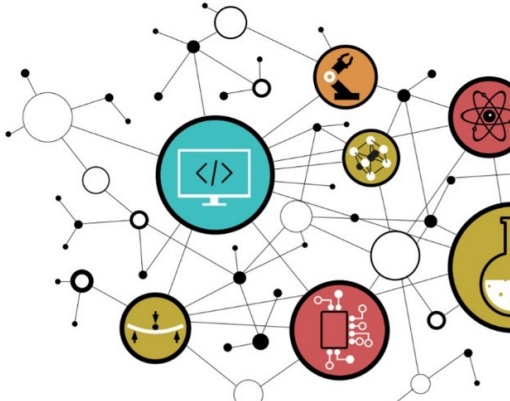
PICSimLab



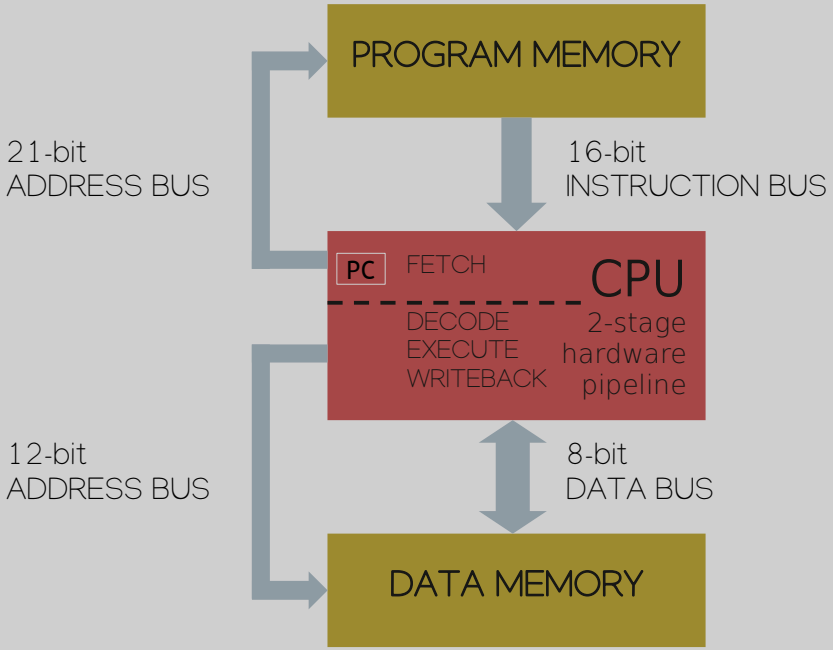
MPLABX Simulator with Debugger

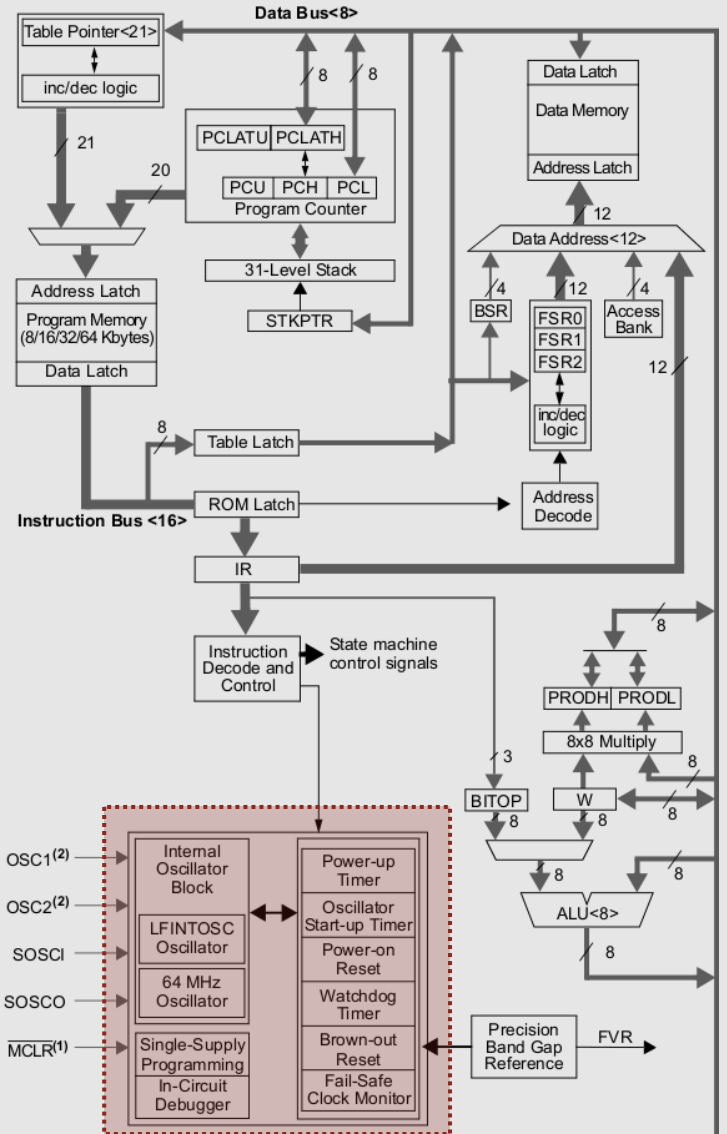


- Insertion ASM dans C
- Allocation statique de variable
- Adressage immédiat
- Instruction de contrôle
- Instructions orientées octet
- Instructions orientées bit
- Solution ASM
- Divers

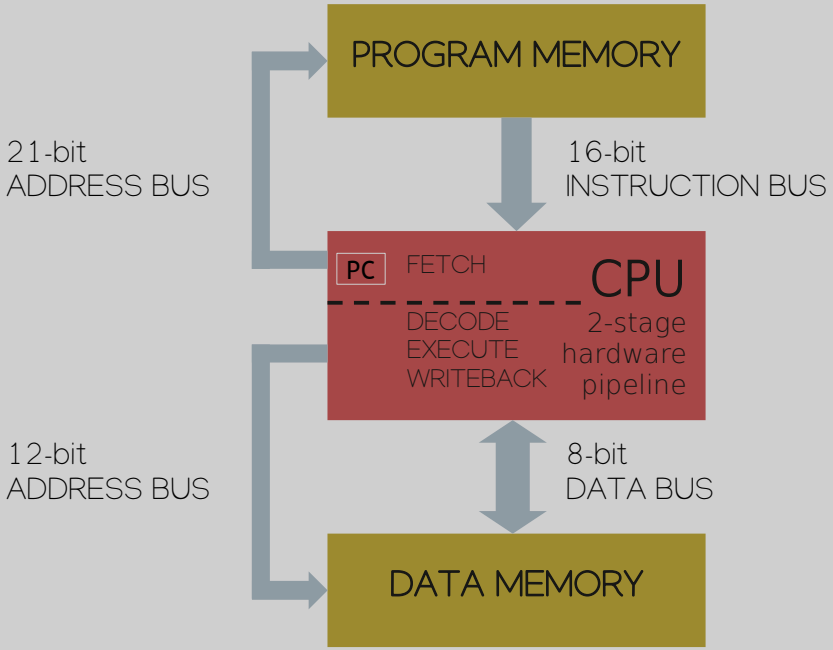


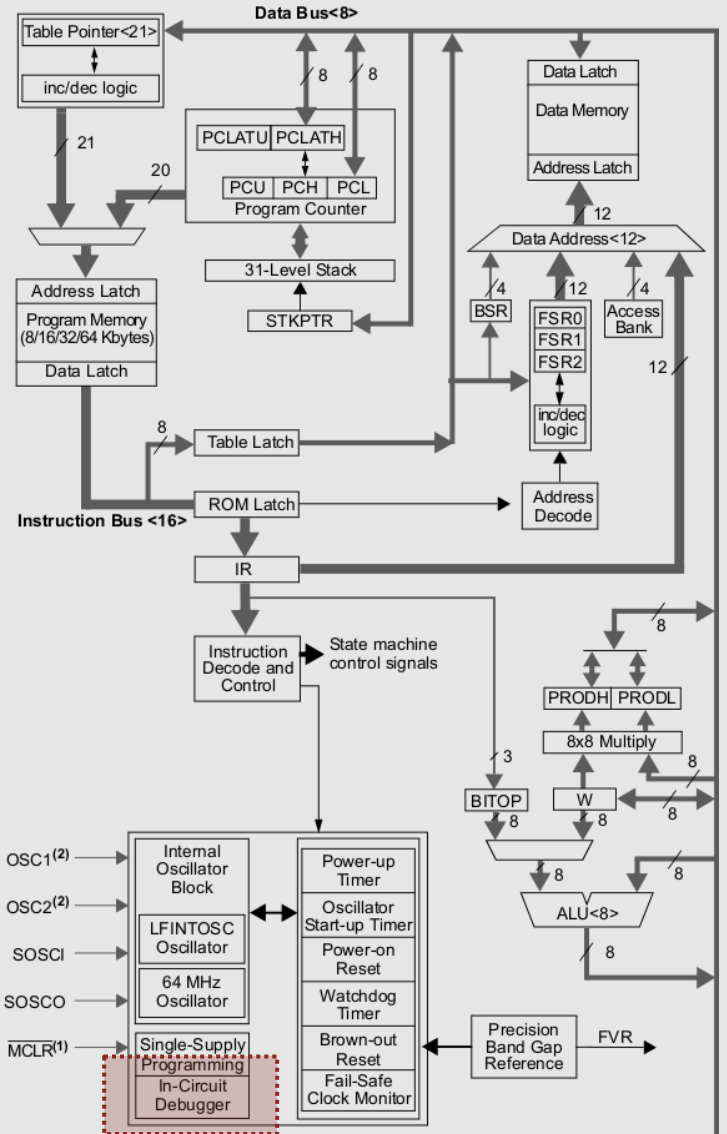
Architecture processeur MCU PIC18





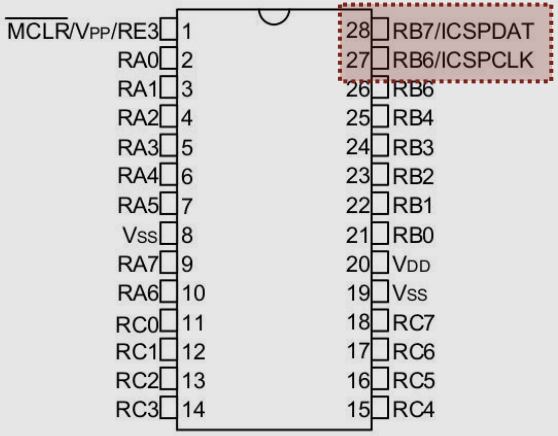
Architecture processeur MCU PIC18



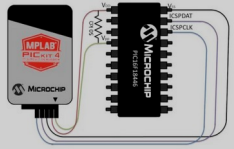


Sans bootloader déjà programmé dans le processeur, nous devons utiliser une sonde JTAG (Join Test Action Group) afin de charger voire debugger le programme depuis l'IDE sur ordinateur vers le MCU cible. Un StarterKit embarque déjà une sonde de programmation à côté du processeur cible de test. Sinon, nous pouvons utiliser des sondes externes plus polyvalentes (ICD4, PICKIT4, etc chez Microchip).

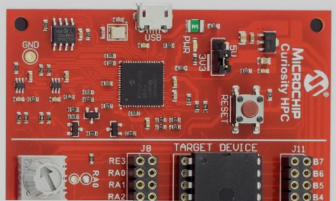
PIC18F27K40
SPDIP 28 pins package

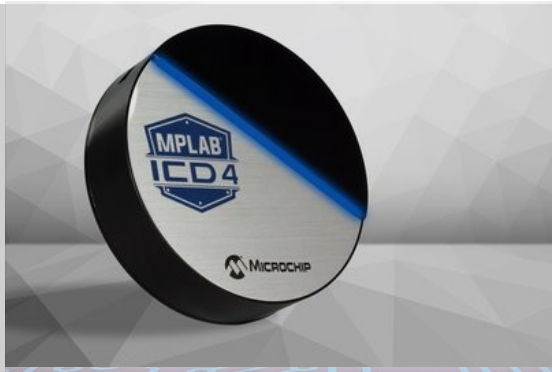


External PICKIT4
JTAG in-circuit programmer/Debugger

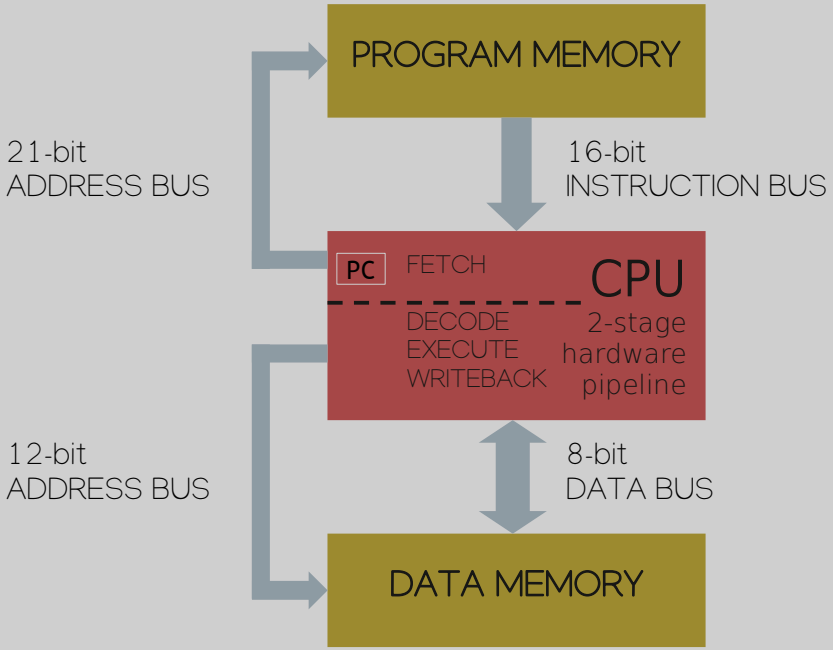


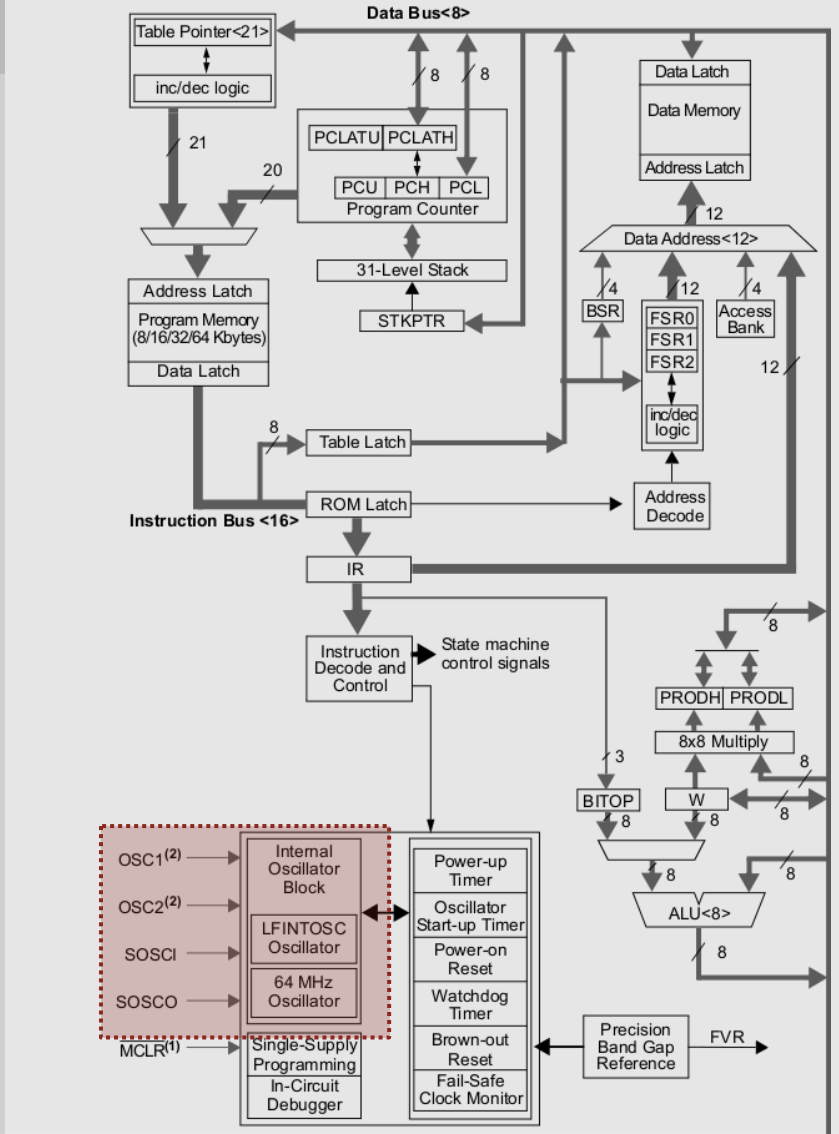
CURIOSITY HPC Starter Kit
with JTAG in-circuit programmer/Debugger



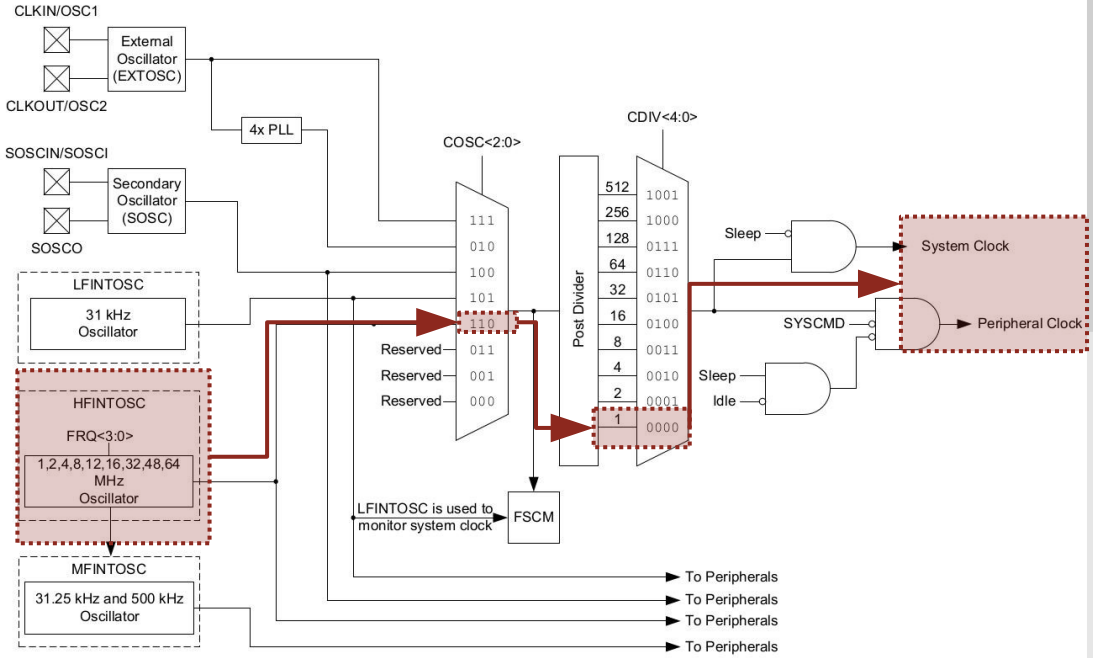


Architecture processeur MCU PIC18





La référence d'horloge du système peut être réalisée par résonateur externe (Quartz) ou interne (MEMS, RC, etc). Les résonateurs externes offrent une meilleure précision (dérive de quelques ppm) mais nécessite un composant supplémentaire sur la carte.



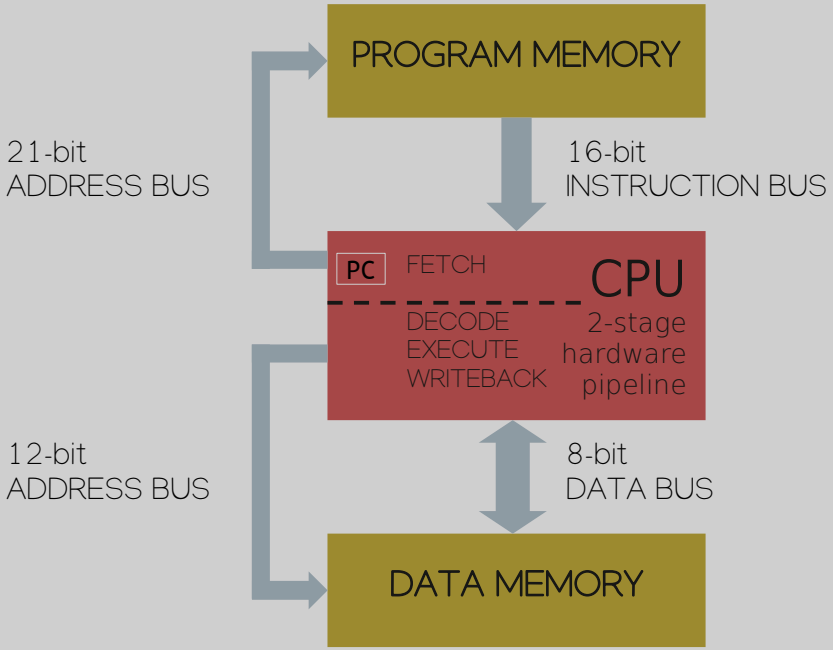
Langage C

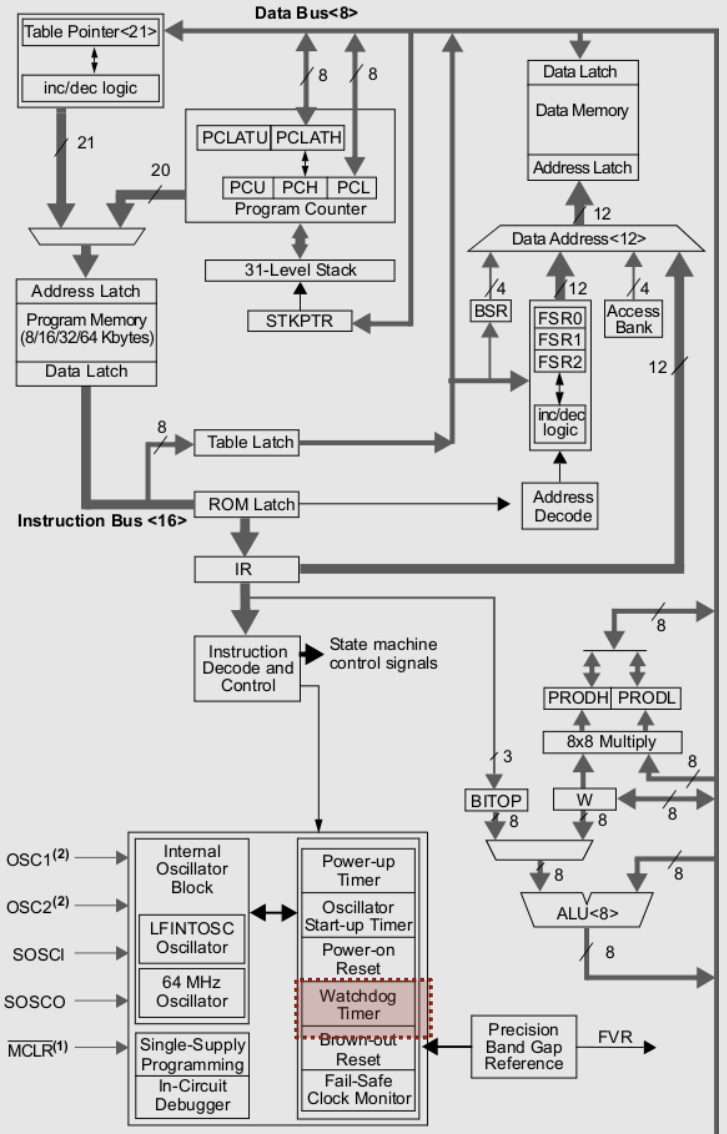
```
#pragma config FEXTOSC = OFF
#pragma config RSTOSC = HFINTOSC_64MHZ
#pragma config MCLRE = EXTMCLR
#pragma config DEBUG = OFF
```

Langage ASM

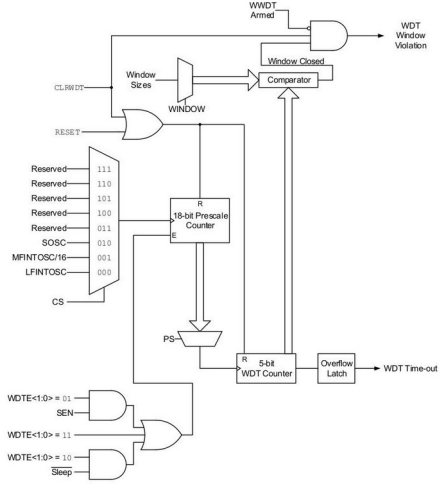
```
CONFIG FEXTOSC = OFF
CONFIG RSTOSC = HFINTOSC_64MHZ
CONFIG MCLRE = EXTMCLR
CONFIG DEBUG = OFF
```

Architecture processeur MCU PIC18



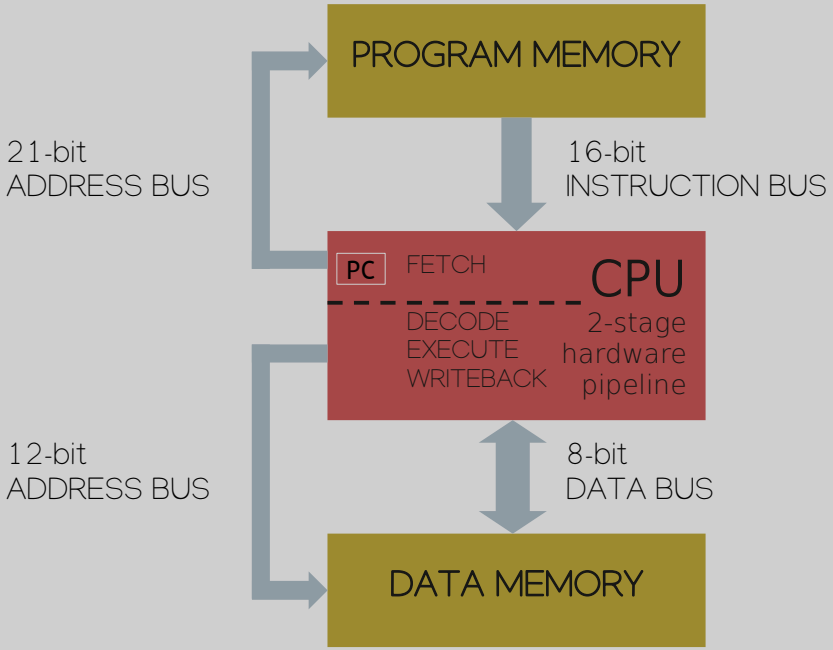


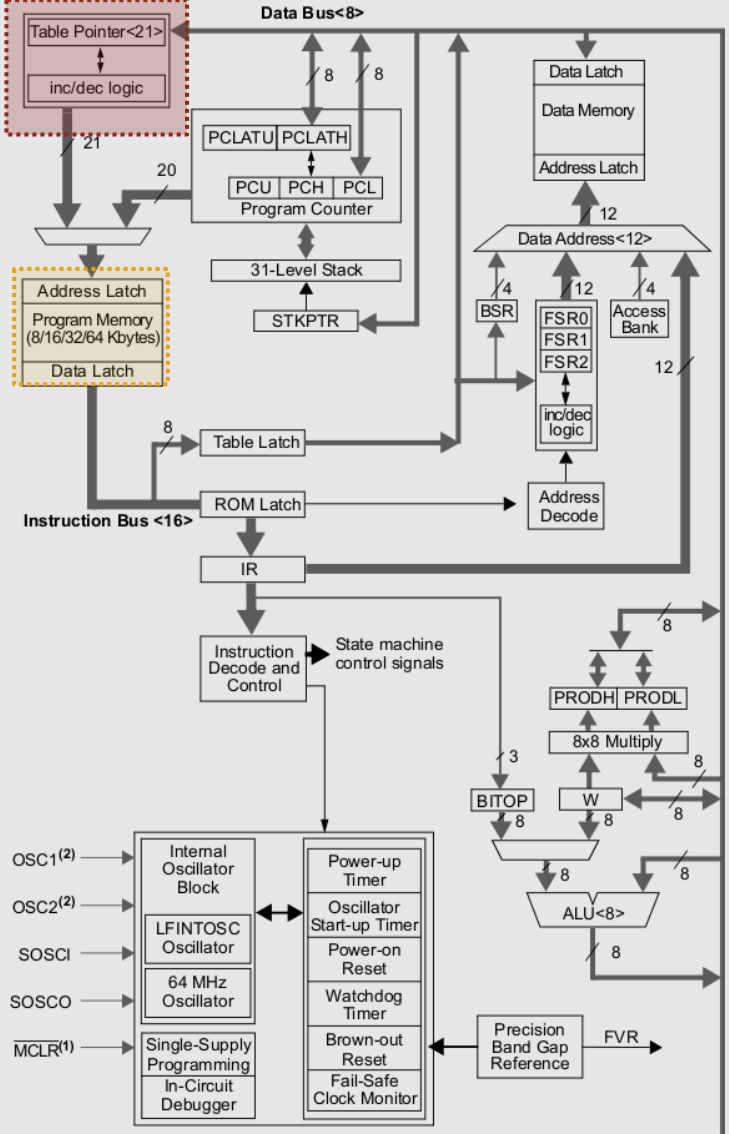
Un watchdog est à ajouter en fin de développement, de test et de validation fonctionnelle d'une application afin d'ajouter une ultime possibilité de redémarrer le programme en cas de défaut grave (application bloquée dans une fonction, boucle infinie, etc). Un Watchdog est un timer pouvant réaliser un RESET (redémarrage) du processeur si il arrive en fin de comptage. Il doit être forcé à zéro par appel de l'instruction CLRWDT en certains endroits clés d'un programme.



Les PIC18 supportent un mode veille et des modes Idle permettant de manager les périphériques activés et ainsi de contrôler la consommation du processeur en phase repos. L'application doit explicitement demander à passer en veille via l'appel de l'instruction SLEEP. Il pourra alors être réveillé par interruption, reset ou par le Watch dog. Une fois en veille, le CPU cesse d'exécuter des instructions mais mémorise néanmoins le contexte d'exécution (registres W, STATUS, BSR, etc) pour le réveil afin de pouvoir restaurer l'état de la machine avant la mise en veille

Architecture processeur MCU PIC18





Les PIC18 offrent une architecture de Harvard et par défaut une faible empreinte de mémoire donnée (application de contrôle). Il est néanmoins possible de manipuler des données chargées en mémoire programme, transformant ainsi l'architecture en processeur de Von Neumann (solution lente). En langage C, utiliser les classes de stockage *rom* ou *ram* (par défaut) afin de forcer les outils à utiliser les instructions associées. Par exemple, *rom char foo* ou *ram char foo/char foo*

DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS																	
TBLRD*	Table Read	2	0000	0000	0000	1000	None	
TBLRD*+	Table Read with post-increment		0000	0000	0000	1001	None	
TBLRD*-	Table Read with post-decrement		0000	0000	0000	1010	None	
TBLRD+*	Table Read with pre-increment		0000	0000	0000	1011	None	
TBLWT*	Table Write	2	0000	0000	0000	1100	None	
TBLWT*+	Table Write with post-increment		0000	0000	0000	1101	None	
TBLWT*-	Table Write with post-decrement		0000	0000	0000	1110	None	
TBLWT+*	Table Write with pre-increment		0000	0000	0000	1111	None	

Microchip propose l'accès gratuit à ses outils de développement, notamment ses chaînes de compilation en version LITE. Ces versions ne permettent pas de lever toutes les options d'optimisation à la compilation. Sous XC8 et C18, les versions payantes permettent notamment d'offrir au compilateur C l'accès aux instructions suivantes

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected
				MSb			LSb	
ADDFSR	f, k	Add literal to FSR	1	1110	1000	ffkk	kkkk	None
ADDULNK	k	Add literal to FSR2 and return	2	1110	1000	11kk	kkkk	None
CALLW		Call subroutine using WREG	2	0000	0000	0001	0100	None
MOVSF	z_s, f_d	Move z_s (source) to 1st word f_d (destination) 2nd word	2	1110 1111	1011 ffff	0zzz ffff	zzzz ffff	None
MOVSS	z_s, z_d	Move z_s (source) to 1st word z_d (destination) 2nd word	2	1110 1111	1011 xxxx	1zzz xzzz	zzzz zzzz	None
PUSHL	k	Store literal at FSR2, decrement FSR2	1	1110	1010	kkkk	kkkk	None
SUBFSR	f, k	Subtract literal from FSR	1	1110	1001	ffkk	kkkk	None
SUBULNK	k	Subtract literal from FSR2 and return	2	1110	1001	11kk	kkkk	None

Merci pour votre attention ...