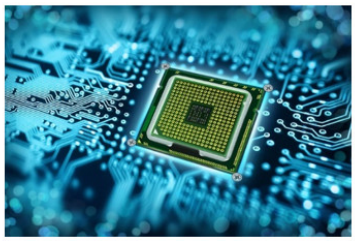
ENSI CAEN

GÉNIE PHYSIQUE ET SYSTÈMES EMBARQUES
1ère année

SYSTÈMES EMBARQUES

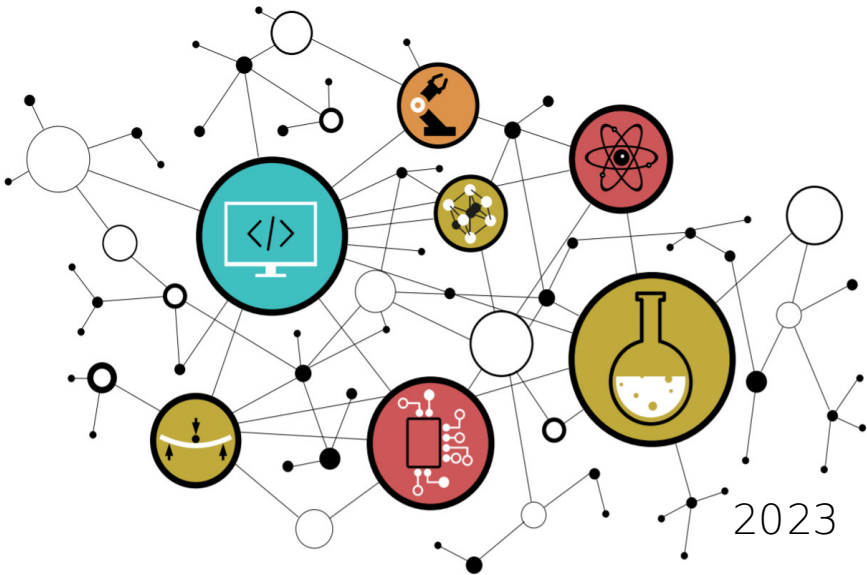
SUPPORT DE COURS

MICROCHIP





2023

CONTACTS

Établissement

ENSICAEN
6 boulevard Maréchal Juin
CS 45053
14050 CAEN cedex 04

Équipe pédagogique

Isabelle Lartigau - TP
isabelle.lartigau@ensicaen.fr

Dimitri Boudier - TP
dimitri.boudier@ensicaen.fr

Arnaud Martin - TP
arnaud.martin@ensicaen.fr

hugo descoubes – TP et COURS
hugo.descoubes@ensicaen.fr
+33 (0)2 31 45 27 61

RESSOURCES

http://foad.ensicaen.fr/

Les différentes ressources numériques sont accessibles sur la platefome pédagogique de l'ENSICAEN (aucune authentification requise, accès libre). Archive de travail mcu.zip. *Ne pas oublier de s'inscrire au cours avant tout dépôt !*

https://foad.ensicaen.fr/course/view.php?id=116

PROGRAMME ET OBJECTIFS

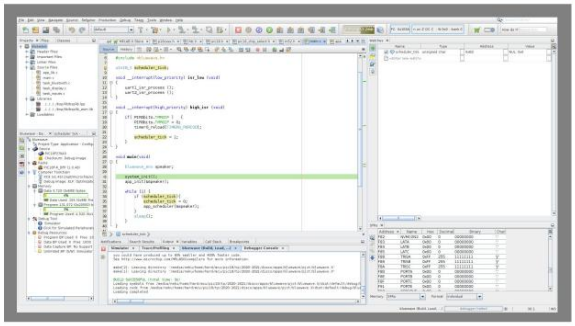
- COURS – *Comprendre le fonctionnement et l'architecture matérielle d'un processeur MCU*

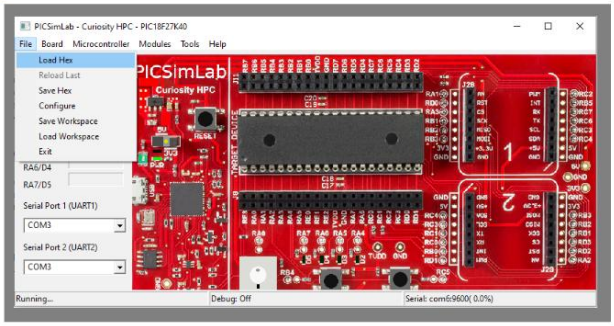
Le cours a pour objectif d'asseoir une bonne compréhension de l'architecture matérielle et du fonctionnement d'un processeur à CPU. Afin d'illustrer et présenter les grands éléments constitutifs de ce type de processeur (CPU, mémoires, périphériques et bus), nous travaillerons sur processeur MCU (Micro Controller Unit ou microcontrôleur). Une fois les grand concepts architecturaux présentés, nous appliquerons nos représentations sur technologie MCU 8bits PIC18 développée par la société Microchip. Nos analyses se feront à l'étage registre du processeur et à l'étage assembleur du modèle de développement logiciel. Une fois l'architecture matérielle assimilée, nous nous intéresserons aux méthodologies de conception et de développement d'application *bare-metal* (sans OS), notamment au concept de *scheduling offline* (stratégie d'ordonnancement). Pour clôturer les phases de cours, nous parcourrons les problématiques liées à des stratégies et techniques de communication rencontrées en Systèmes Embarqués (Liaison série, SPI, I2C, etc).

- TRAVAUX PRATIQUES – *Développer, tester, valider et documenter une solution logicielle embarquée sur MCU. Développement de bibliothèques pilotes (drivers), d'applications de test unitaires et d'une application produit.*

Pour une grande partie des réalisations, nous aurons à développer un *BSP* (Board Support Package) ou *HAL* (Hardware Abstraction Layer) *from scratch* (en partant de rien) et en travaillant à l'étage registre du processeur (plus bas niveau de développement sur machine). En résumé, nous allons tout faire de A à Z. Dans notre cas, le BSP doit être vu comme une collection de fonctions logicielles pilotes (drivers) assurant le contrôle des fonctions matérielles périphériques internes (GPIO, Timer, UART, etc) voire externes (module Bluetooth, afficheur alphanumérique LCD, etc). Notre BSP sera dédié à notre processeur et notre carte. Une migration de technologie ou de solution (processeur et/ou carte) nécessiterait ajustement et redéveloppement. Une fois le BSP développé, testé, validé, documenté et la bibliothèque statique générée, nous développerons une application audio Bluetooth *bare-metal* (sans OS) l'utilisant. L'application implémentera notamment un *scheduler offline*. Nous ne pourrons alors qu'imaginer l'infini potentiel créatif s'ouvrant devant nos yeux !

OUTILS DE DÉVELOPPEMENT





Suivre les indications présentent dans la section OUTILS DE DEVELOPPEMENT sous l'espace moodle associé à l'enseignement afin d'installer les outils. Les outils de développement proposés par Microchip sont libres d'utilisation du moment que nous utilisons les versions dîtes Free ou Lite (outils sans options d'optimisation). Chaque exercice de TP peut être pré-compilé voire testé en simulation à la maison avant l'arrivée en séance. De même, l'installation des outils puis l'utilisation en mode simulation pour une analyse de traduction de programmes C vers ASM PIC18 est sans aucun doute l'une des solutions d'apprentissage et de révision les plus efficaces lorsque nous n'avons pas en possession les plateformes matérielles. Voici ci-dessous la synthèse des outils à installer :

- IDE (Integrated Development Environment) MPLABX v5.50 : https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive

- Toolchain C XC8 v1.45 (Free Mode) : https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive

- Terminal asynchrone de communication TeraTerm (dernière version) : https://ttssh2.osdn.jp/index.html.en

- Drivers VCP (Virtual COM Port) pour chip USB to UART de FTDI (dernière version) : http://www.ftdichip.com/Drivers/VCP.htm

- Simulateur PICSimLab pour Windows 64bits : https://foad.ensicaen.fr/mod/resource/view.php?id=24874

- Emulateur Null Modem com0com (dernière version) : https://sourceforge.net/projects/com0com/files/com0com/3.0.0.0/com0com-3.0.0.0-i386-and-x64-signed.zip/download

ÉVALUATIONS DES COMPETENCES

L'enseignement comportera 2 évaluations distinctes. Une évaluation sur table et une évaluation pratique. Voici le détail des points sur lesquels vous serez évalués :

ÉVALUATION SUR TABLE - 2h30 : *Feuille manuscrite A4 recto/verso*

- CONNAÎTRE – 6pts : Questions de culture générale pouvant traiter sur tout point aborde en séance de cours présentiel ou présent dans le support de travail. *Connaissances fondamentales et culture scientifique de l'ingénieur électronicien*

- COMPRENDRE – 10pts : Exercice de traduction d'un programme C vers un équivalent en assembleur PIC18. Niveau d'exigence proche de l'exercice réalisé en cours. *Comprendre et maîtriser le travail d'un processeur numérique et des outils de compilation*

- ANALYSER – 4pts : Analyse d'un programme réalisant une application simple sur une architecture processeur non découverte en enseignement. *Adaptabilité de l'ingénieur aux concepts étudiés sur de nouvelles technologies*

ÉVALUATION PRATIQUE - 1h30 : *Tous documents autorisés*

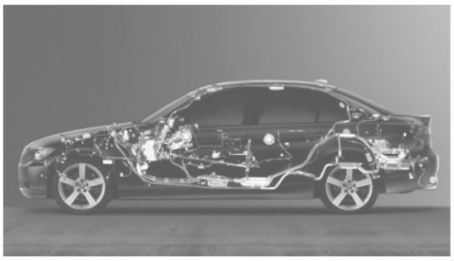
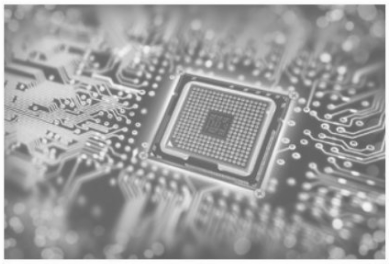
- DEVELOPPER – 20pts : Réalisation d'un projet simple sur matériel réel pouvant traiter sur tout point abordé durant les séance de Travaux Pratiques. Se référer à son référent de TP pour les questions relatives à cette évaluation. *Faculté de l'ingénieur à répliquer ses compétences opérationnelles sur un cahier des charges nouveau*

SOMMAIRE DU POLYCOPIE DE COURS

CARTE CURIOSITY HPC

SYSTÈMES EMBARQUES

CARTE CURIOSITY



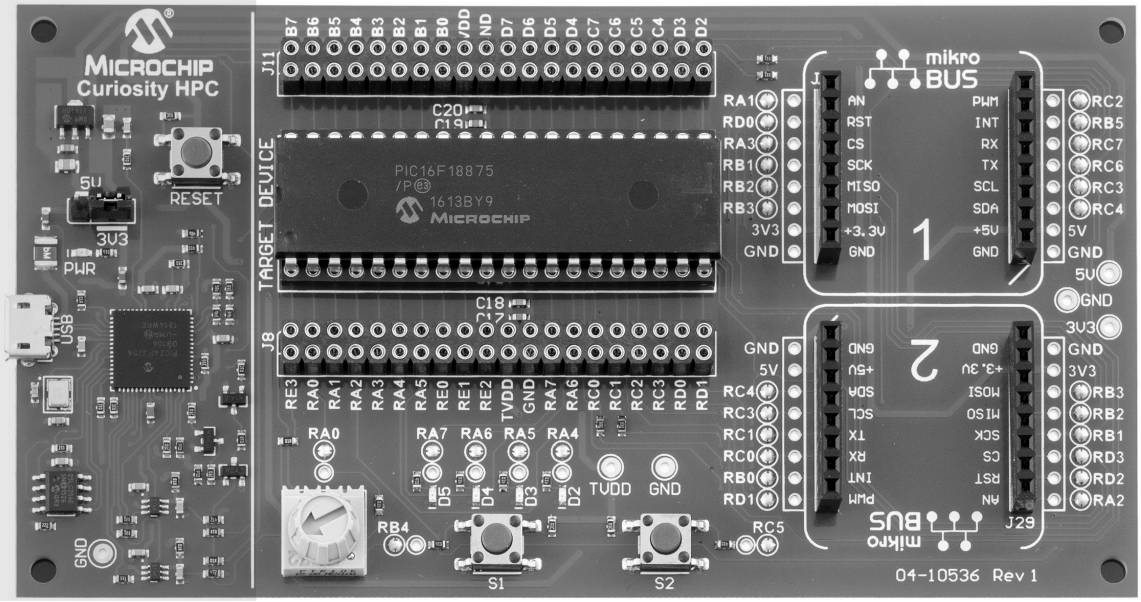



neku - hugo descoubes – enseignant Systèmes Embarqués – ENSICAEN - France
GNU\Linux Ubuntu 20.04 LTS – LibreOffice 6.4.6.2 – 2022

SYSTÈMES EMBARQUES
Starter Kit Curiosity HPC

Programmer/Debugger

Application

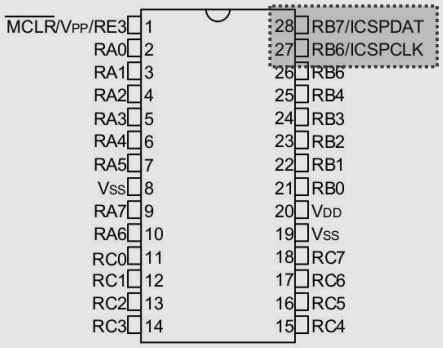


https://www.microchipdeveloper.com/boards:curiosityhpc

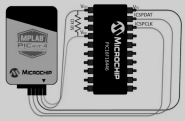
SYSTÈMES EMBARQUES
JTAG in-circuit Programmer/Debugger

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).

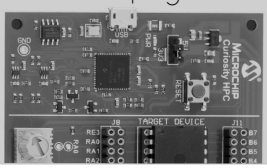
PIC18F27K40
SPDIP 28 pins package



External PICKIT4
JTAG in-circuit programmer/Debugger



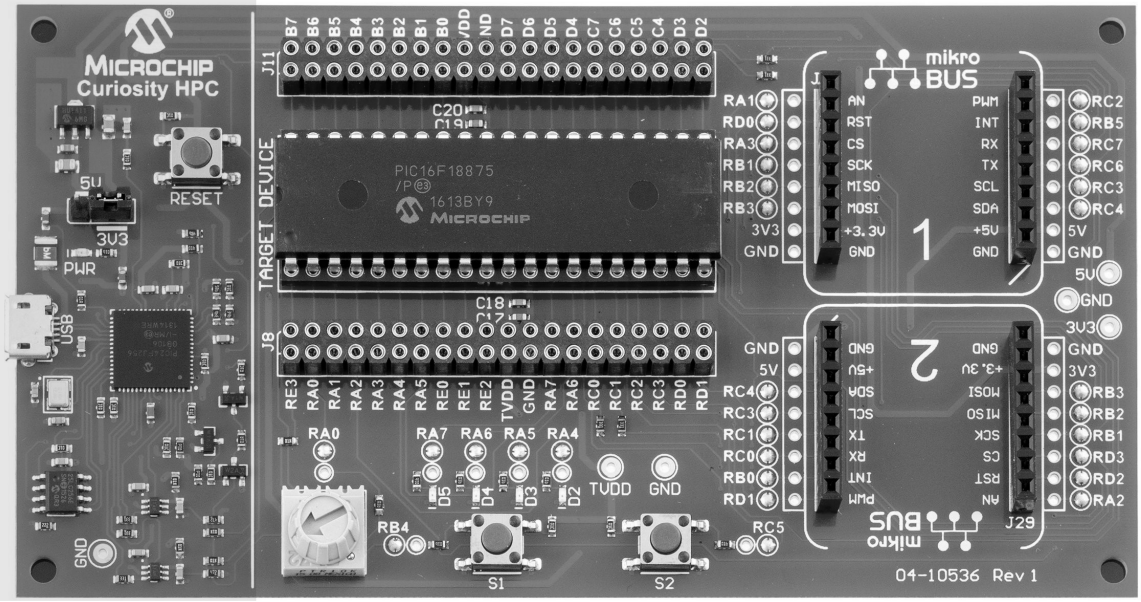
CURIOSITY HPC Starter Kit
with JTAG in-circuit programmer/Debugger



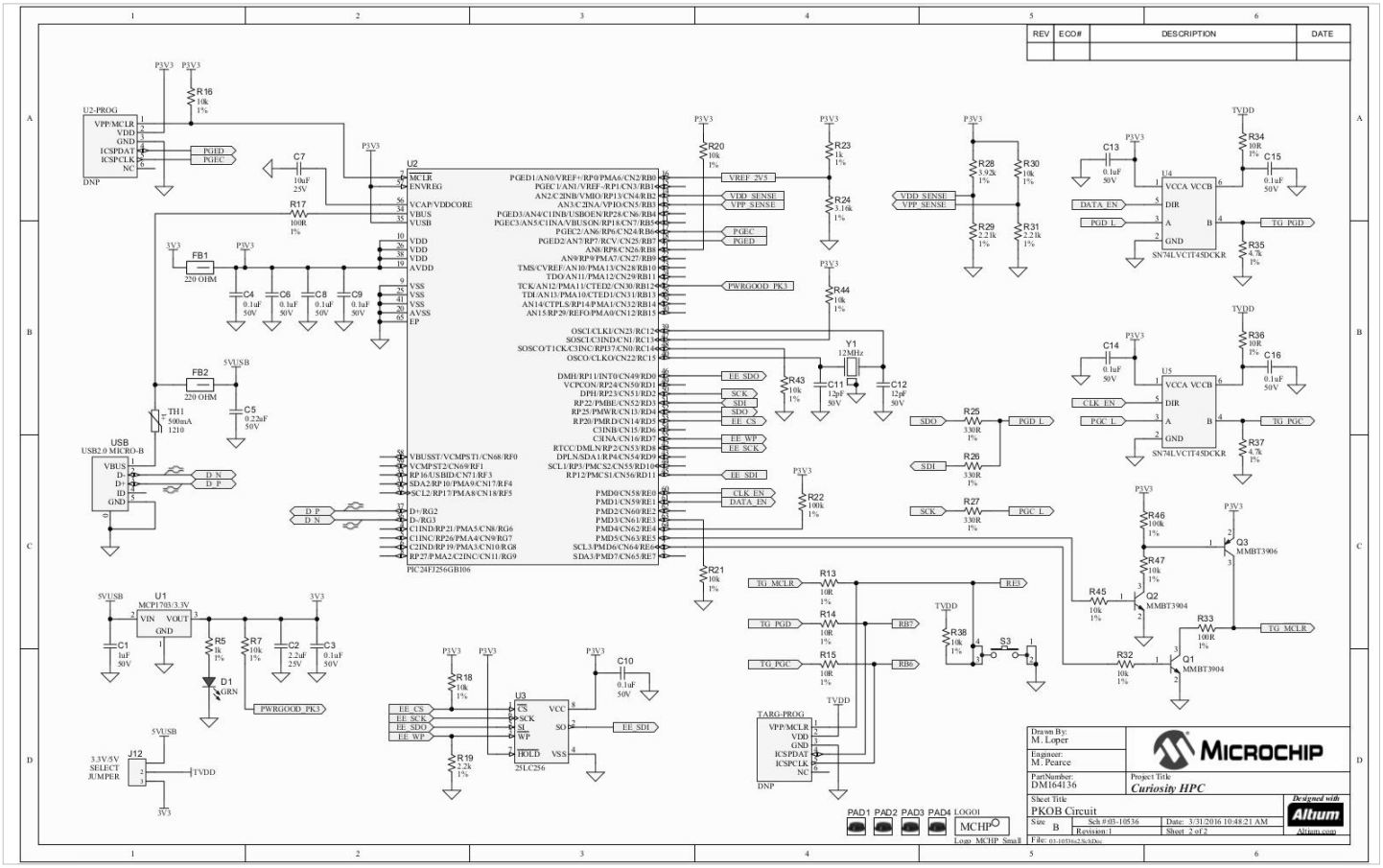
SYSTÈMES EMBARQUES
Starter Kit Curiosity HPC

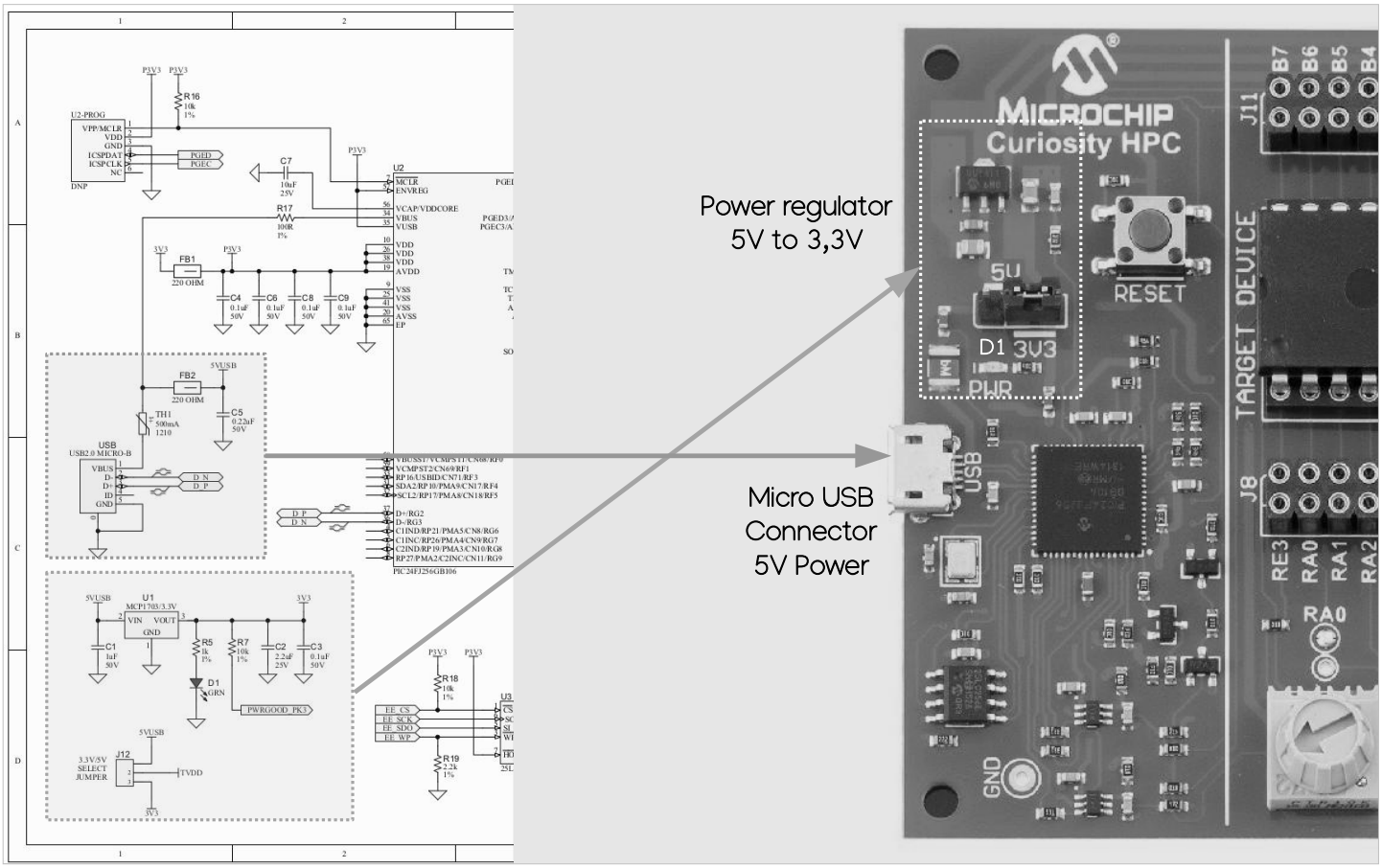
Programmer/Debugger

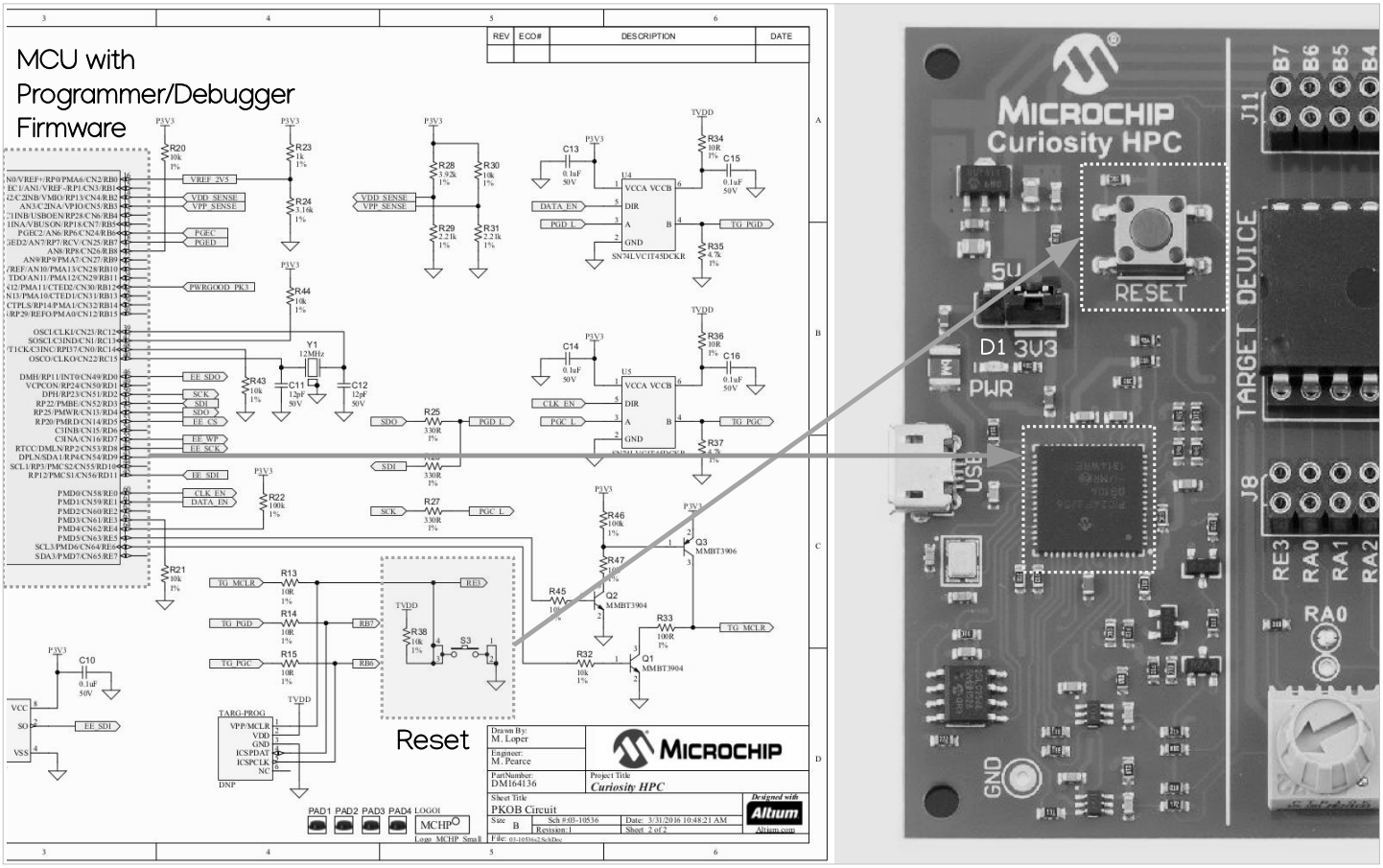
Application



https://www.microchipdeveloper.com/boards:curiosityhpc

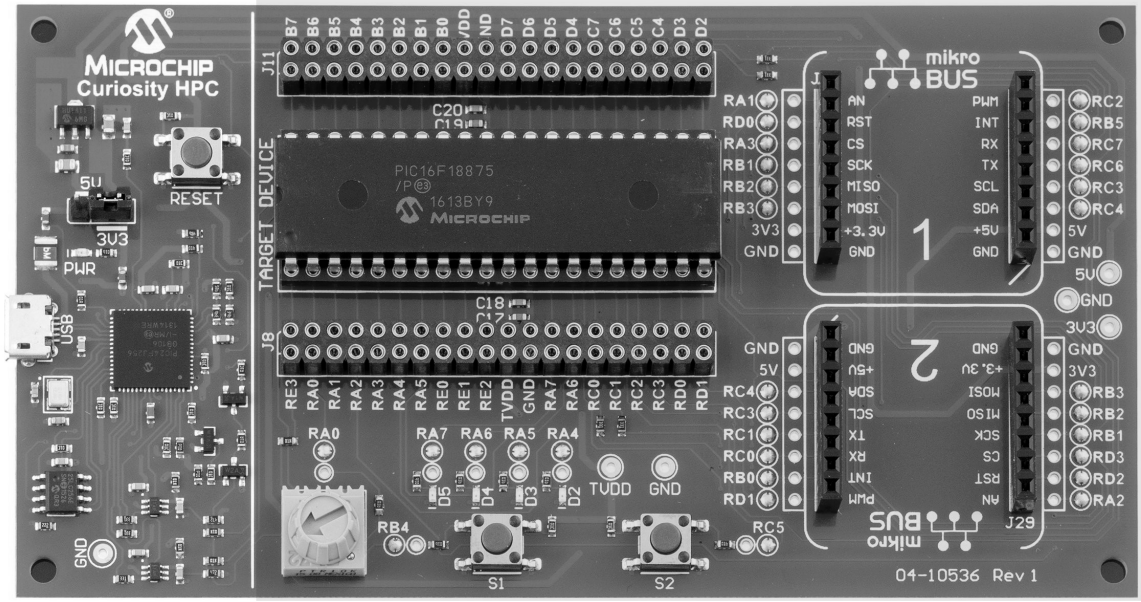

REV
ECO#
DESCRIPTION
DATE
U2-PROG
U2
PIC24FJ256GB106
USB
USB2.0 MICRO-B
U1
MCP1703/3.3V
J12
3.3V/5V SELECT JUMPER
U3
25LC256
U4
SN74LVCH4SDCKR
U5
SN74LVCH4SDCKR
Q1
MMBT3904
Q2
MMBT3904
Q3
MMBT3906
TARG-PROG
DNP
Drawn By:
M. Loper
Engineer:
M. Pearce
PartNumber:
DM164136
Project Title
Curiosity HPC
Sheet Title
PKOB Circuit
Size B
Sch #03-10536
Revision:1
Date: 3/31/2016 10:48:21 AM
Sheet 2 of 2
MICROCHIP
Designed with Altium
Altium.com


Power regulator
5V to 3,3V
Micro USB
Connector
5V Power
MICROCHIP
Curiosity HPC
RESET
5V
D1 3V3
PWR
USB
J11
B7
B6
B5
B4
TARGET DEVICE
J8
RE3
RA0
RA1
RA2
RA0
GND

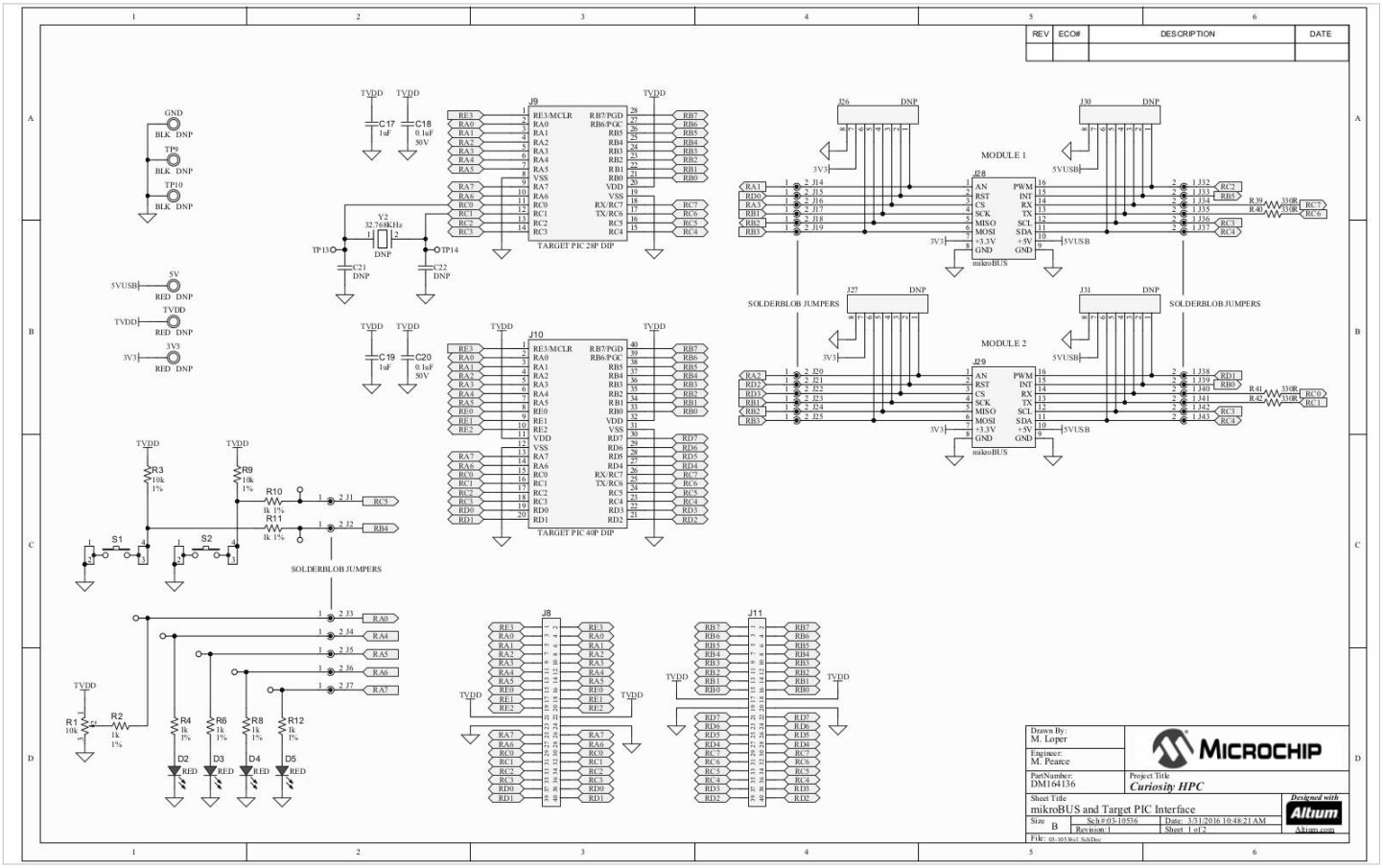


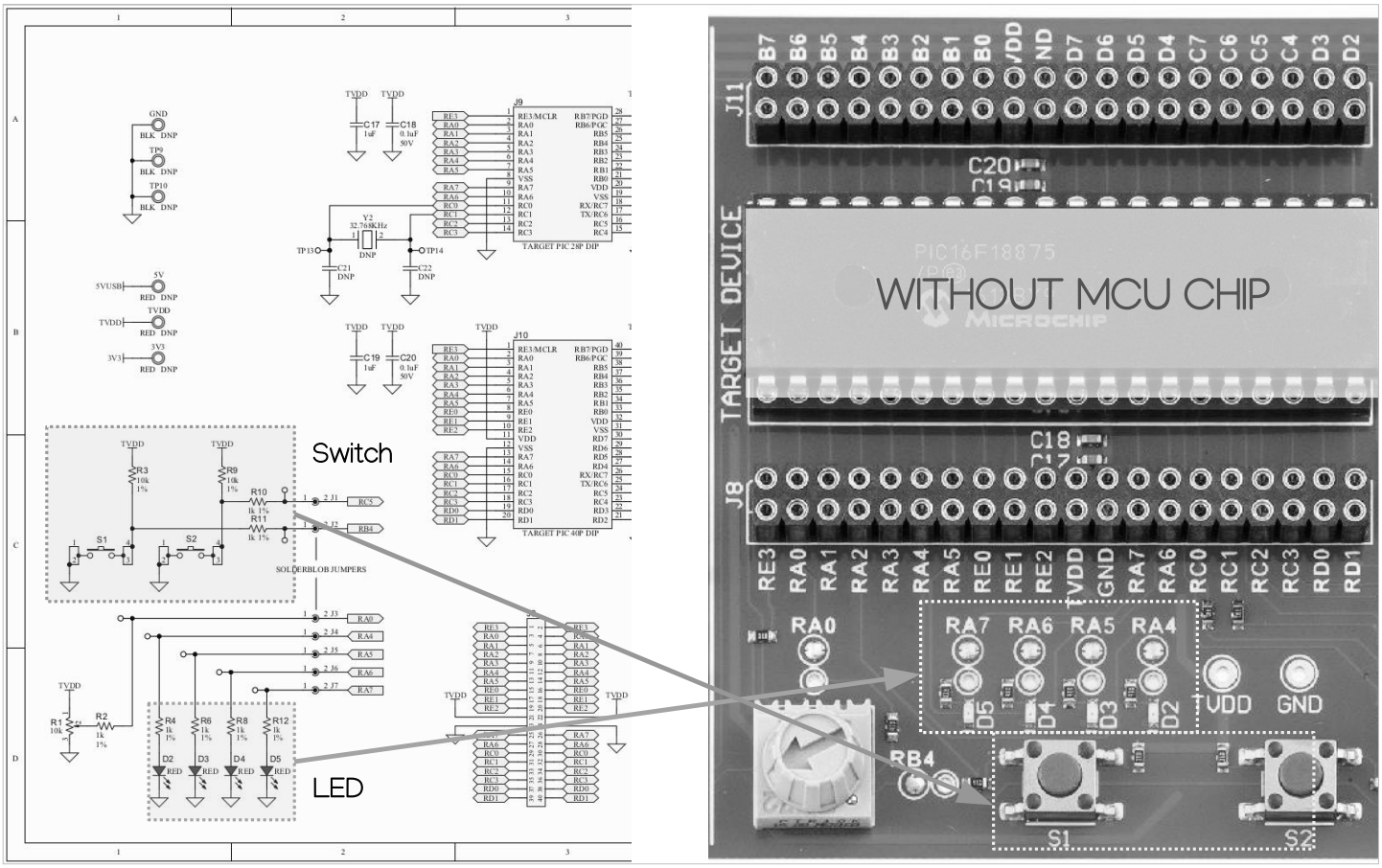
Programmer/Debugger

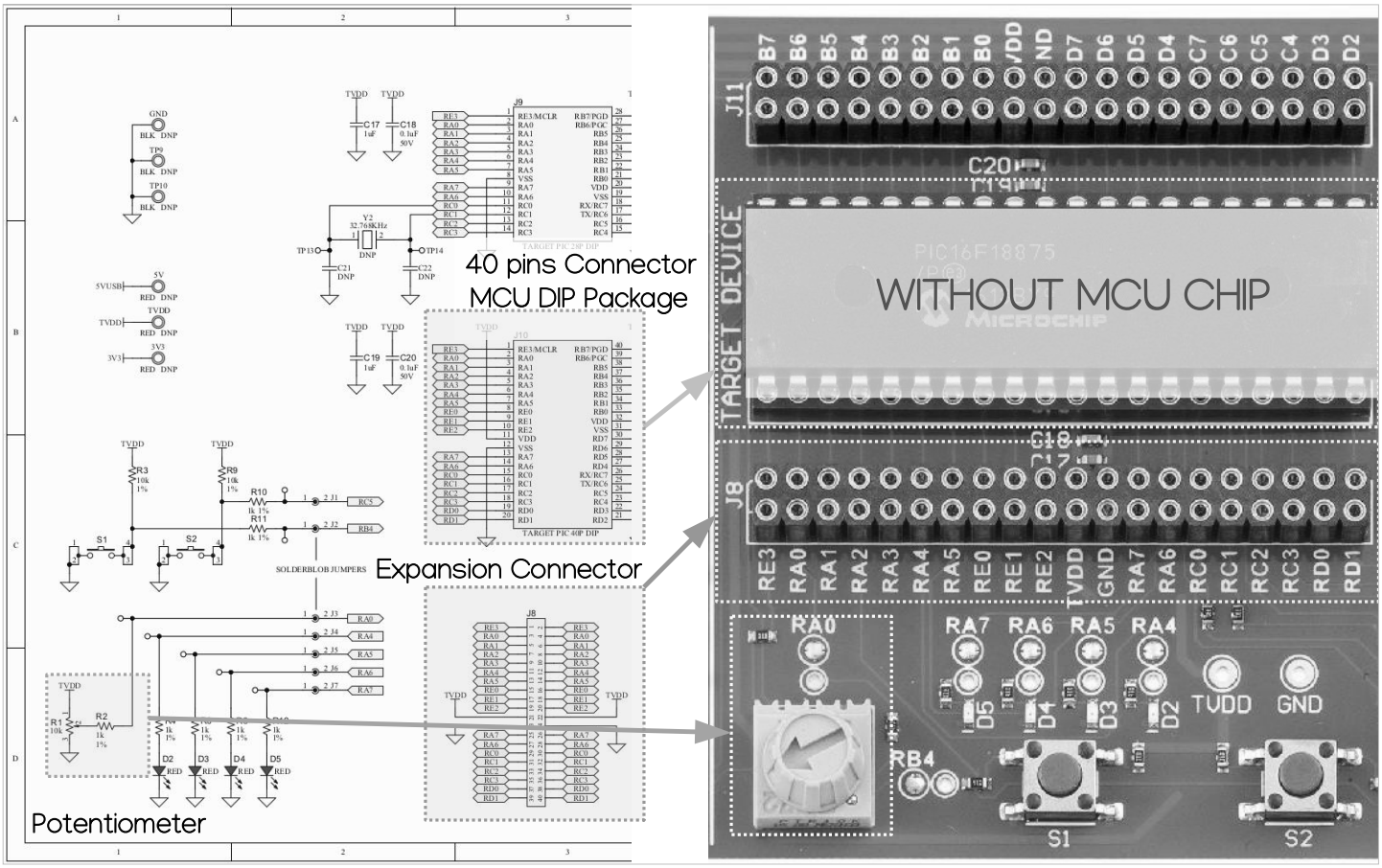
Application

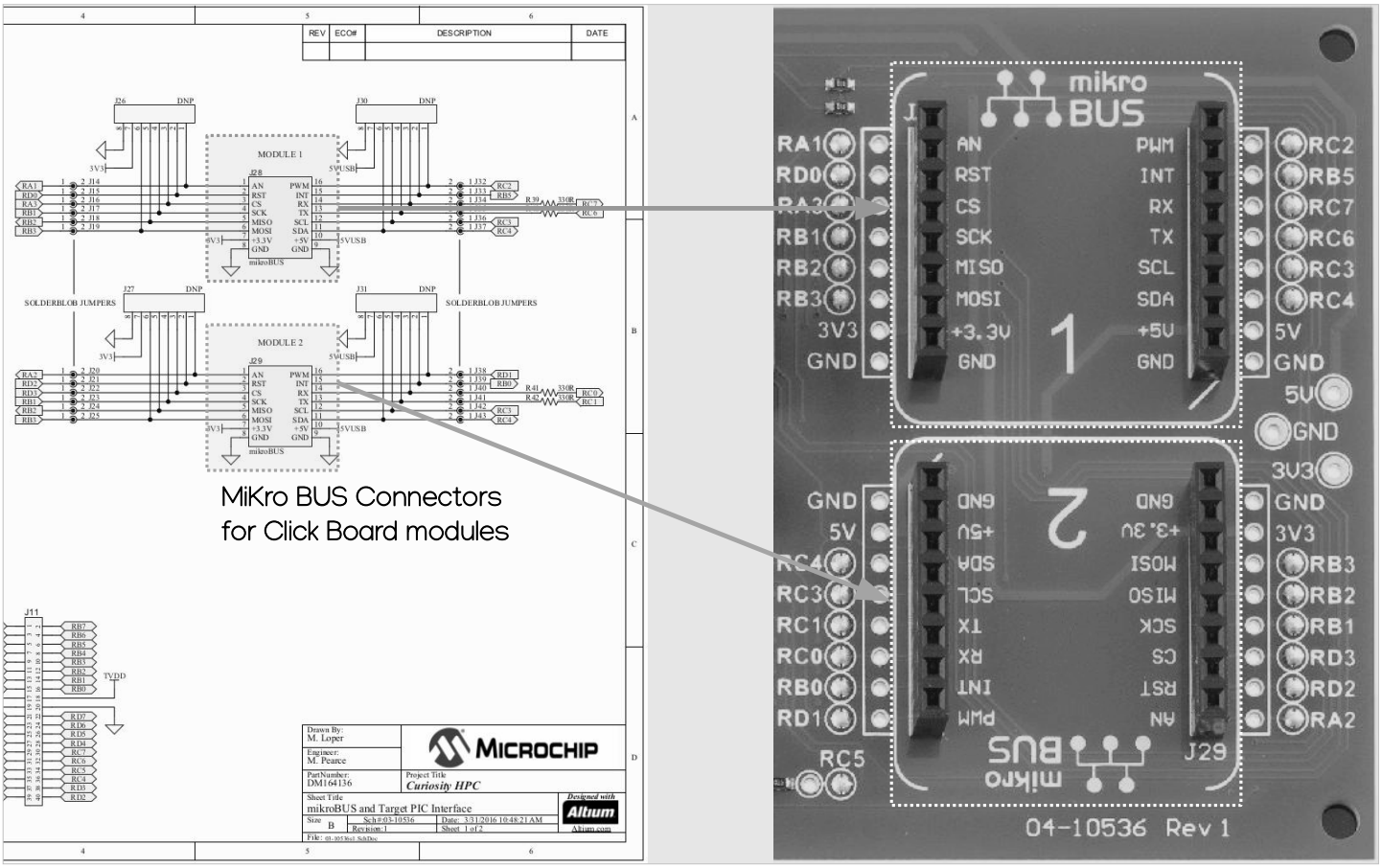


https://www.microchipdeveloper.com/boards:curiosityhpc


REV
ECO#
DESCRIPTION
DATE
J9
TARGET PIC 28P DIP
J10
TARGET PIC 40P DIP
J26
J30
J27
J31
MODULE 1
MODULE 2
J28
J29
mikroBUS
SOLDERBLOB JUMPERS
J8
J11
Drawn By:
M. Loper
Engineer:
M. Pearce
PartNumber:
DM164136
Project Title
Curiosity HPC
Sheet Title
mikroBUS and Target PIC Interface
Size
B
Sch#:03-10536
Revision:1
Date: 3/31/2016 10:48:21 AM
Sheet 1 of 2
MICROCHIP
Altium

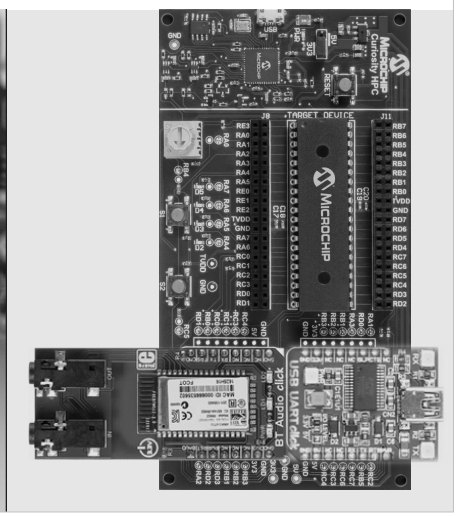

Switch
LED
J9
J10
J8
J11
TARGET PIC 28P DIP
TARGET PIC 40P DIP
SOLDERBLOB JUMPERS
WITHOUT MCU CHIP
TARGET DEVICE
J11
J8
RA0
RA7
RA6
RA5
RA4
VDD
GND
RB4
S1
S2


40 pins Connector
MCU DIP Package
Expansion Connector
Potentiometer
WITHOUT MCU CHIP
TARGET DEVICE
J11
J8
RA0
RA7 RA6 RA5 RA4
TVDD GND
RB4
S1
S2


MiKro BUS Connectors
for Click Board modules
MODULE 1
MODULE 2
SOLDERBLOB JUMPERS
mikro BUS
04-10536 Rev 1

SYSTÈMES EMBARQUES
Prototypage avec modules Click Board de MikroElectronika

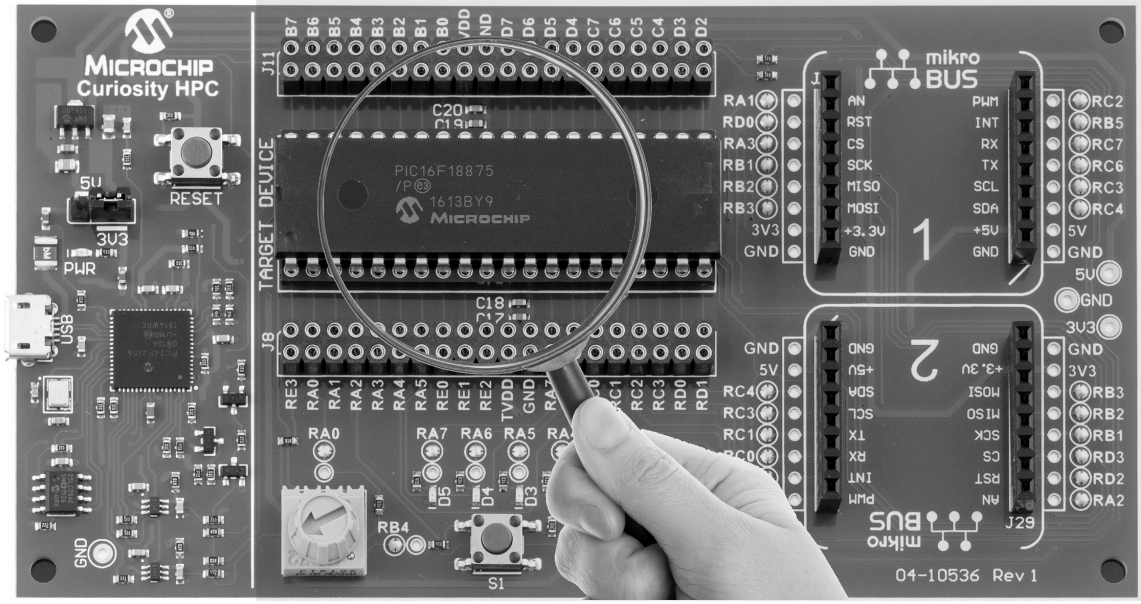
La carte Curiosity HPC possède deux connecteurs mikroBUS permettent également d'ajouter des modules externes Click Board proposés par la société Mikroelektronika. Des centaines de modules externes sont actuellement disponibles en catalogue (Bluetooth, audio, WIFI, contrôle de moteur, afficheurs LCD, capteurs divers, etc)



SYSTÈMES EMBARQUES
Starter Kit Curiosity HPC

Programmer/Debugger

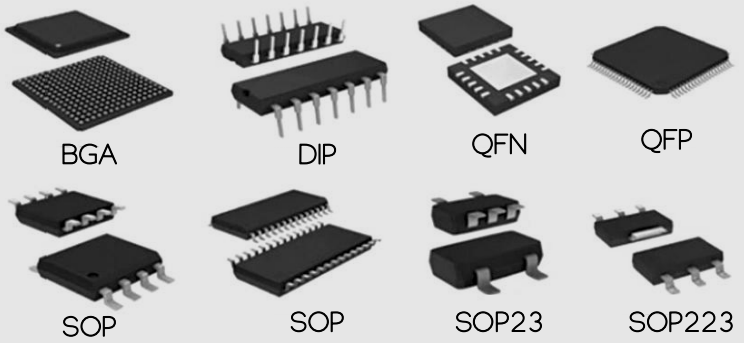
Application

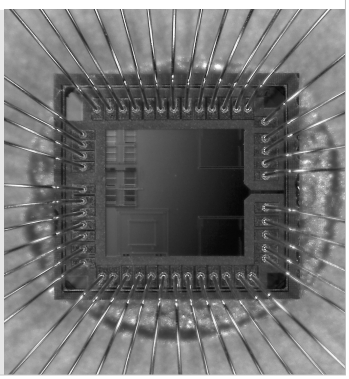


https://www.microchipdeveloper.com/boards:curiosityhpc

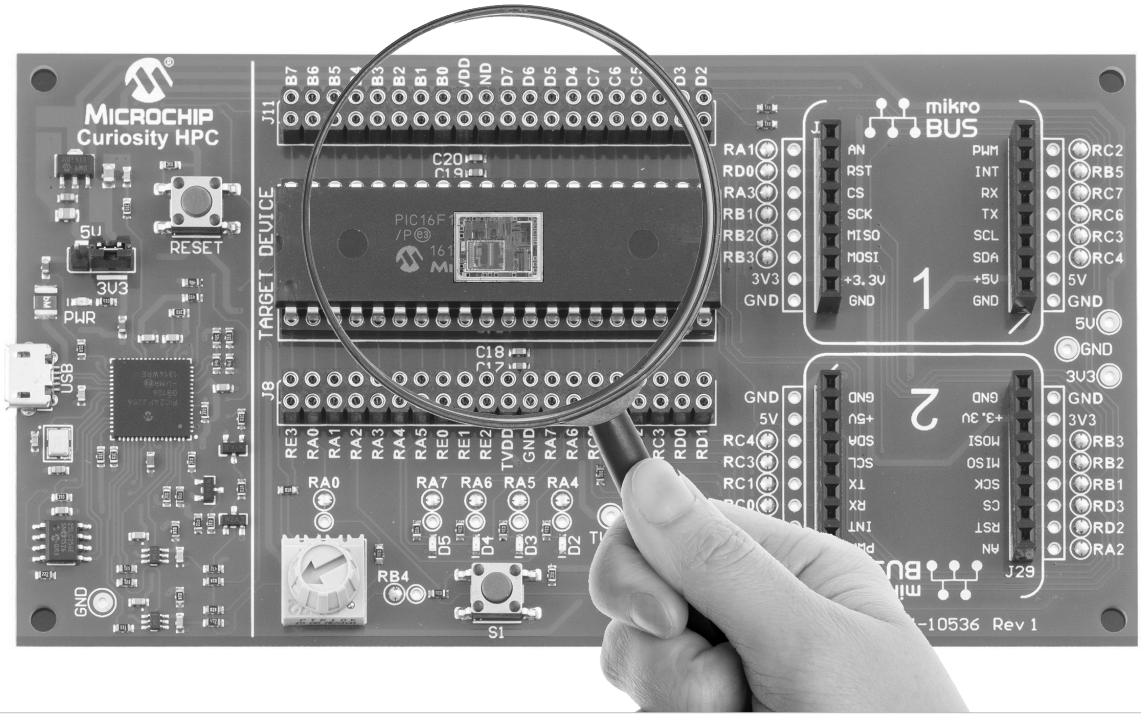
SYSTÈMES EMBARQUES
Types de boîtiers pour composants

Comme la plupart des composants électronique, un MCU peut être décliné en plusieurs boîtiers ou packages par le fabricant (DIP, BGA, QFN, SOP, etc). Chaque boîtier offrant en général un ensemble d'avantages et d'inconvénients. Il est à noter que seul le boîtier change (encombrement, accessibilité des broches, dissipation thermique, prototypage ou production, techniques de dépose, etc). La puce de silicium embarquée est la même. La curiosity HPC, en tant que matériel de prototypage, est elle dédiée aux boîtiers DIP 28 ou 40 broches.





SYSTÈMES EMBARQUES
A la découverte des MCU PIC18 de Microchip ...



ARCHITECTURE PIC18

Chapter 4

Microchip PIC18 Architecture

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

2021-2022

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Market shares of silicon manufacturers and MCU suppliers in 2021.

2Q21 Top 10 Semiconductor Sales Leaders ($M, Including Foundries)

2Q21 Rank	1Q21 Rank	Company	Headquarters	1Q21 Total IC	1Q21 Total O-S-D	1Q21 Total Semi	2Q21 Total IC	2Q21 Total O-S-D	2Q21 Total Semi	2Q21/1Q21 % Change
1	2	Samsung	South Korea	16,152	920	17,072	19,262	1,035	20,297	19%
2	1	Intel	U.S.	18,676	0	18,676	19,304	0	19,304	3%
3	3	TSMC (1)	Taiwan	12,911	0	12,911	13,315	0	13,315	3%
4	4	SK Hynix	South Korea	7,270	358	7,628	8,762	451	9,213	21%
5	5	Micron	U.S.	6,629	0	6,629	7,681	0	7,681	16%
6	6	Qualcomm (2)	U.S.	6,281	0	6,281	6,472	0	6,472	3%
7	8	Nvidia (2)	U.S.	4,842	0	4,842	5,540	0	5,540	14%
8	7	Broadcom Inc. (2)	U.S.	4,364	485	4,849	4,400	490	4,890	1%
9	10	MediaTek (2)	Taiwan	3,849	0	3,849	4,496	0	4,496	17%
10	9	TI	U.S.	3,793	235	4,028	4,030	269	4,299	7%
—	—	Top-10 Total		84,767	1,998	86,765	93,262	2,245	95,507	10%

(1) Foundry (2) Fabless

Source: Company reports, IC Insights' *Strategic Reviews* database

Leading MCU Suppliers ($M)

2021 Rank	Company	Headquarters	2020	2021	21/20 % Chg	2021 Marketshare
1	NXP	Europe	2,980	3,795	27%	18.8%
2	Microchip	U.S.	2,872	3,584	25%	17.8%
3	Renesas	Japan	2,748	3,420	24%	17.0%
4	ST	Europe	2,506	3,374	35%	16.7%
5	Infineon	Europe	1,953	2,378	22%	11.8%

Source: Company reports, IC Insights

Design Only

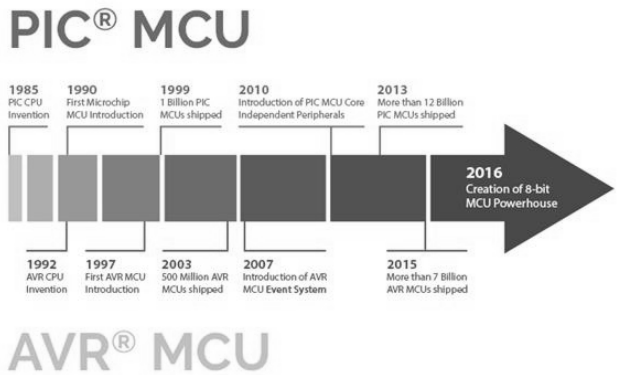
Manufacturing Only

MICROCHIP

Technology used during labs

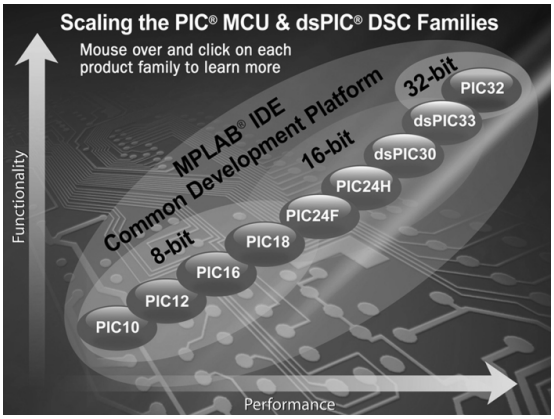
The American company Microchip is an electronic device manufacturer. Most of its turnover (fr: *chiffre d'affaires*) is due to MCUs: about 60% come from the PIC family according to Microchip ESC Filing.

In 2016 Microchip bought Atmel, its major concurrent on the 8-bit MCU market.

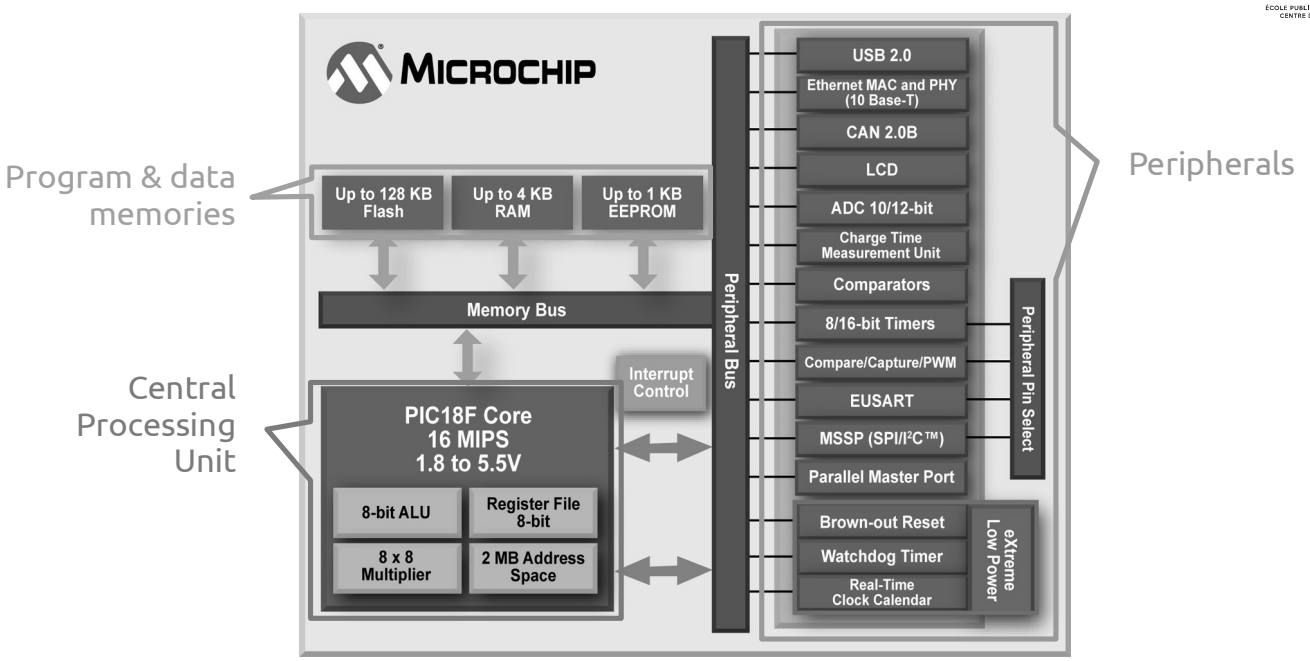


With its large range of MCU solutions, Microchip can win its clients loyalty by offering them the possibility to aim for various applications and markets.

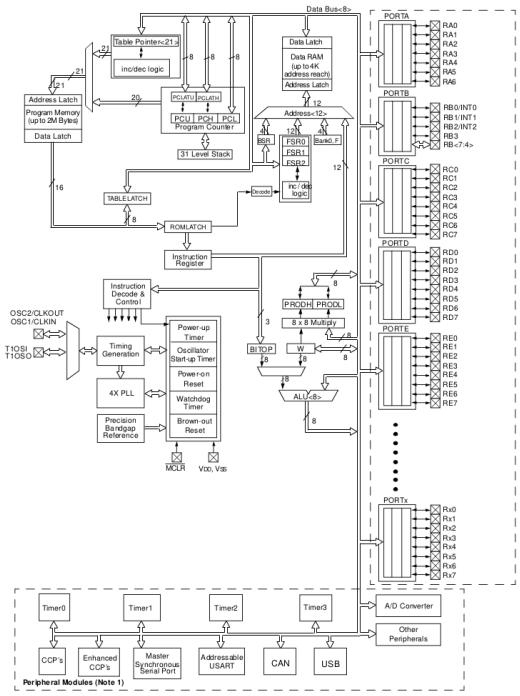
Like many manufacturers, Microchip also supplies tools that make it easy to switch from a specific architecture to another (e.g. migration from PIC18 to PIC32).



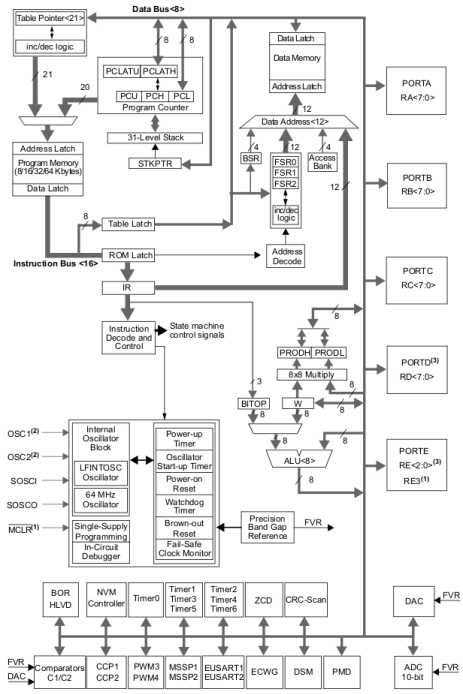
MICROCHIP PIC18 MCUs



MICROCHIP PIC18 MCUs



Spot the differences!



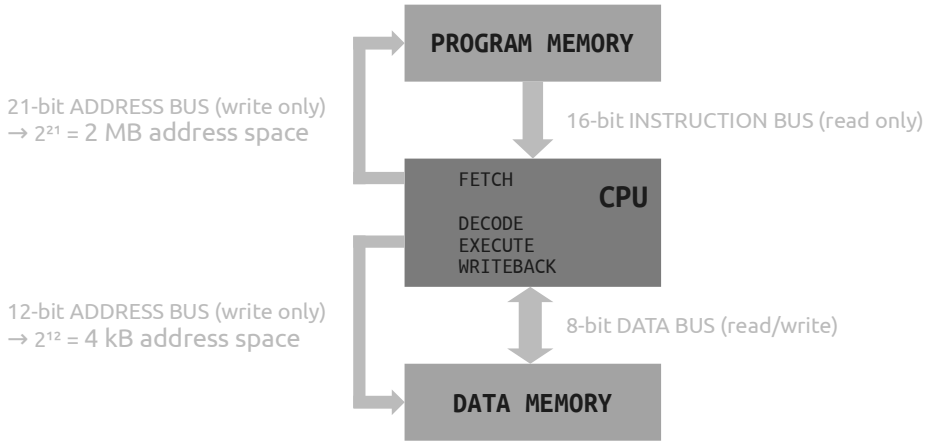
PIC18C family (left)

PIC18F27/47K40 (right)

CENTRAL PROCESSING UNIT

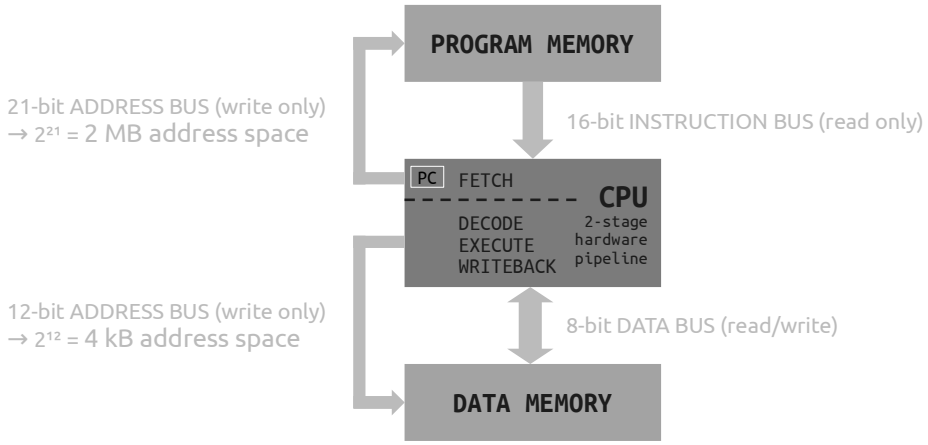
ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Just like the Atmel AVR, Microchip PIC18 follow a Harvard architecture: program and data memories are physically separated and their own addressing space are distinct.



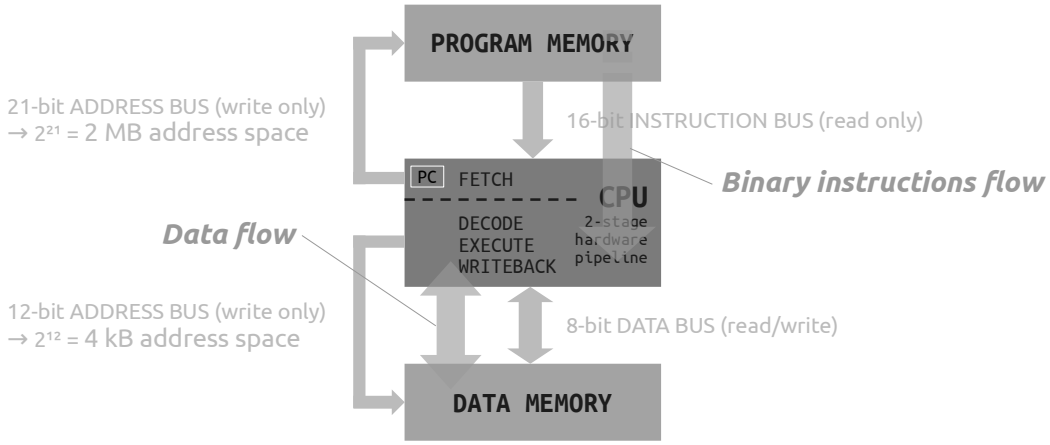
PIC18 CPUs are designed with a 2-stages hardware pipeline. They can decode-execute-store an instruction while fetching the next one from the program memory.

Max performance of 16 MIPS.



Except in sleep mode, the CPU executes a constant flow of instructions coming out of the main memory.

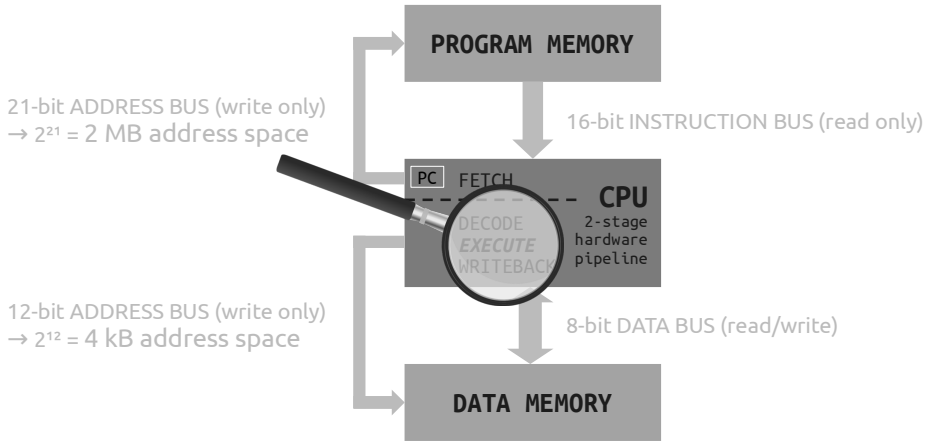
Some instructions ask the CPU to load or store a data from or to the data memory.



Architecture

Let's dive into the EXECUTION stage to see the PIC18 Execution Units (EUs).

As the PIC18 is a 8-bit MCU, the EUs can only operate on 8-bit integer values.



Execution Unit: ALU

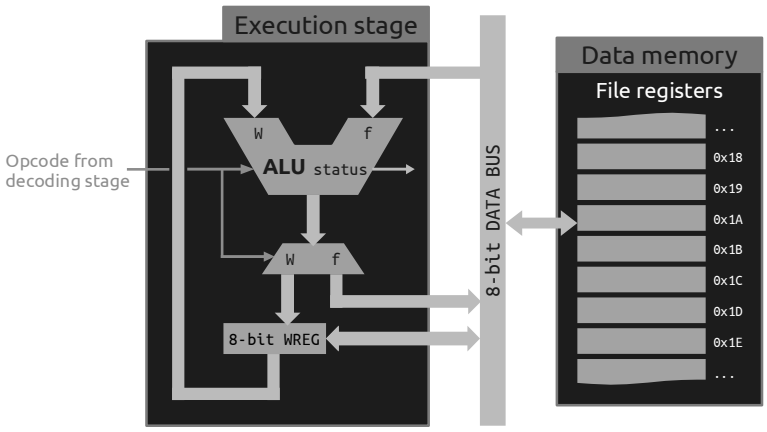
The **Arithmetic and Logic Unit (ALU)** is an execution unit in charge of arithmetic (+, -) and logic (&, |, ^, !, …) operations on 8-bit integer values.

Arithmetic operations

```
ADDWF
INCF
SUBWF
DECF
...
```

Logic operations

```
ANDWF
IORWF
XORWF
CLRF
SETF
...
```



Execution Unit: ALU

Any arithmetic or logic operation use by default a first 8-bit operand stored in the working register WREG and a second operand in the file register (data memory).

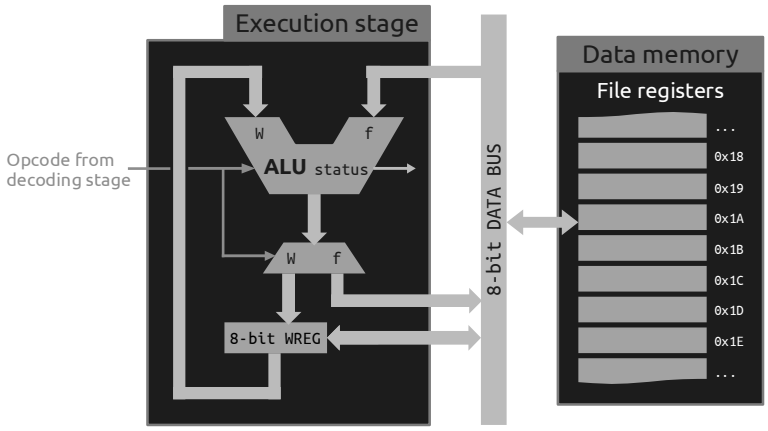
Example of an assembly language instruction and equivalent operation code (opcode).

PIC18 assembly language:

```
ADDWF f, d, a
```

16-bit opcode:

```
0010 01da ffff ffff
```



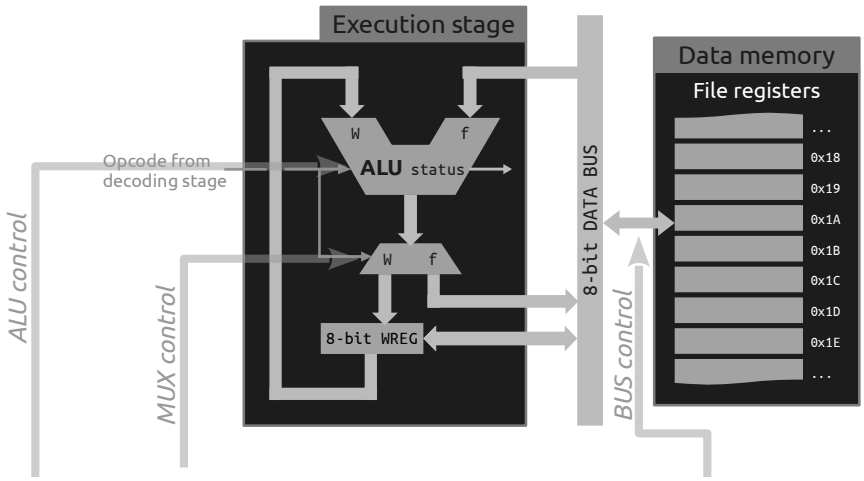
Execution Unit and Decoding Unit

PIC18 assembly language:

```
ADDWF f, d, a
```

16-bit opcode:

```
0010 01da ffff ffff
```



16-bit opcode:	0010 01	d	a	ffff ffff
	Opcode Unique for each instruction	**Destination** d=0 → Work register d=1 → file register	**Access bank** a=0 → Access bank a=1 → All banks, BSR	**Source/Dest. address** 8-bit, Relative to a bank

PIC18 CENTRAL PROCESSING UNIT

Execution Unit: 8-bit x 8-bit integer multiplication

Numeration

```
uint8 * uint8 = uint16
 int8 *  int8 =  int15
```

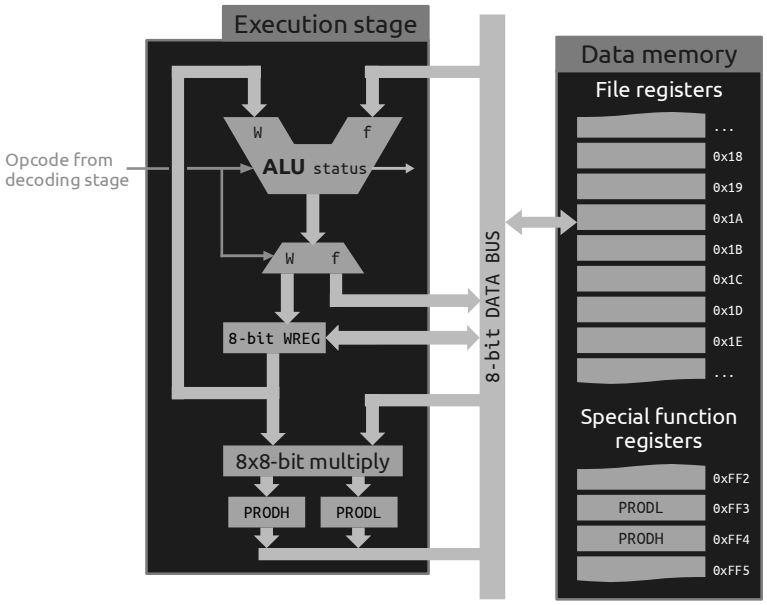
PIC18 multiply operations

MULWF (W-reg to F-reg)
MULLW (Litteral to W-Reg)

C and asm example

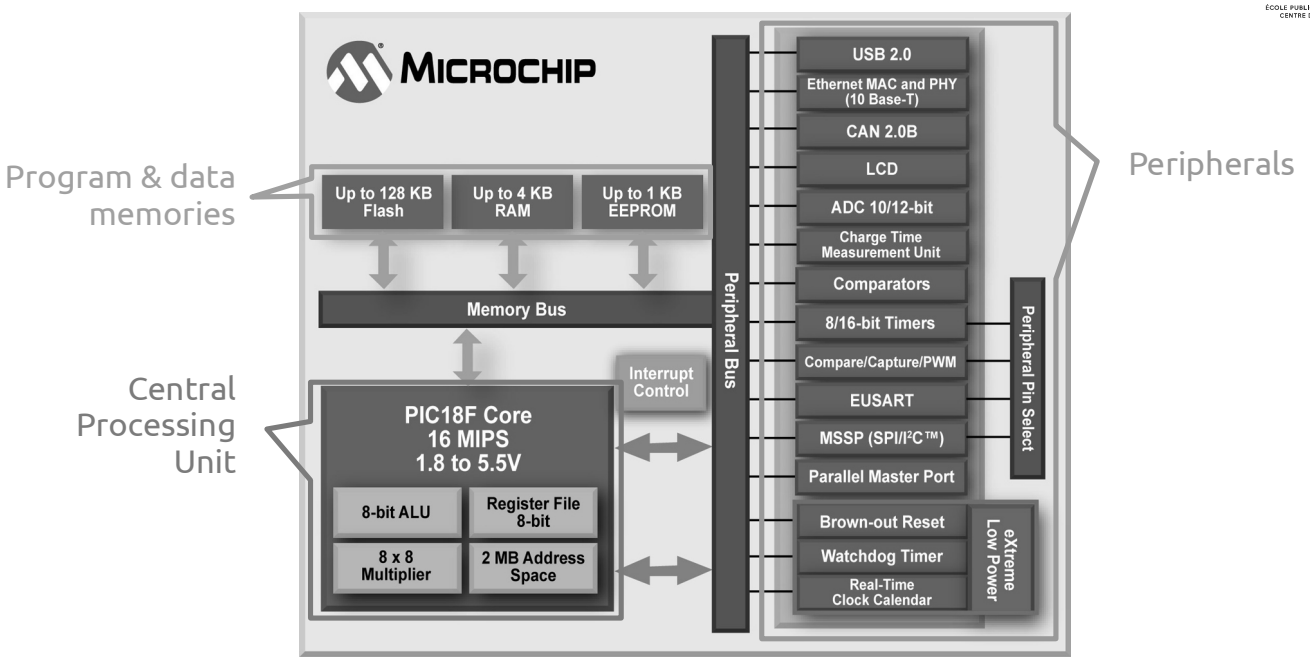
```
static short foo;
foo = 3*7;

…
MOVLW         3
MULLW         7
MOVFF         PRODL, <foo_L_12bit_address>
MOVFF         PRODH, <foo_H_12bit_address>

      or
MOVFF         0xFF3, <foo_L_12bit_address>
MOVFF         0xFF4, <foo_H_12bit_address>
```

PRODL is an alias for 0xFF3, declared in a header
PRODH is an alias for 0xFF4, declared in a header



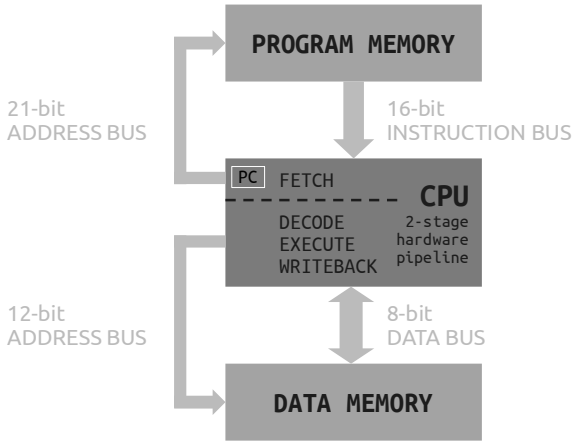
PIC18 CENTRAL PROCESSING UNIR

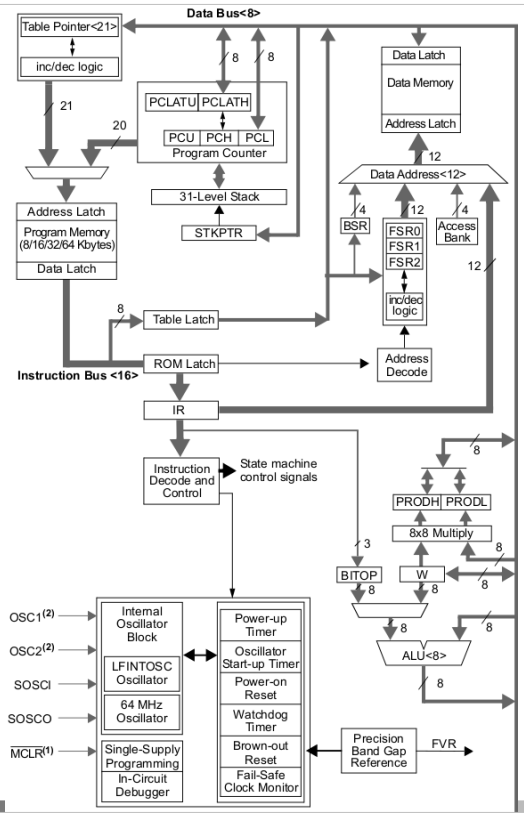
PIC18 architecture



PIC18 CENTRAL PROCESSING UNIT

PIC18F27/47K40 CPU architecture





PIC18 CENTRAL PROCESSING UNIT

PIC18F27/47K40 CPU architecture

Find following items in this schematic

Flash memory

RAM memory

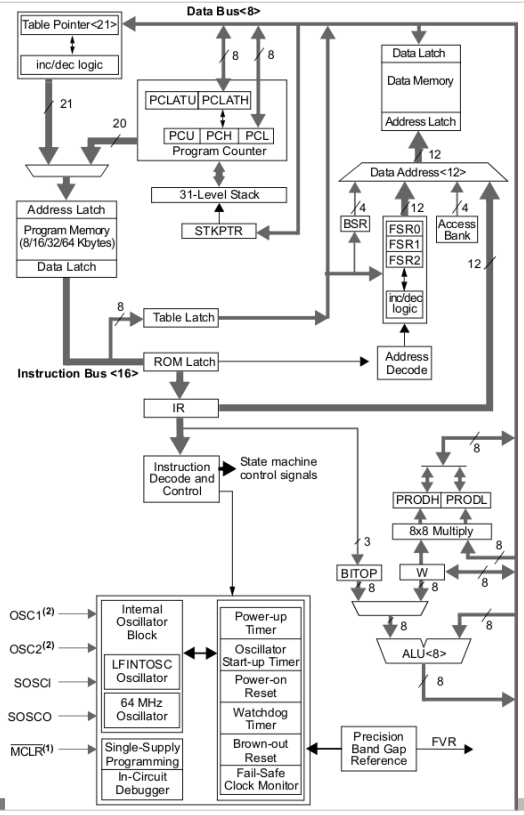
Buses

- program memory address bus
- data memory address bus
- data bus

Hardware pipeline stages

- Fetch
- Decode
- Execute (ALU, multiplier)
- Writeback

Program Counter register



Now that you know how the PIC18 CPU is made, you shall adapt your programming habits.

This CPU (like most of low-power MCUs) does not have any Floating-Point Unit. Therefore you should avoid using floats and doubles, and use integers instead.

Also as this MCU uses an 8-bit CPU you should use 8-bit integers (char C-type) as much as possible. They are usually large enough for control applications.

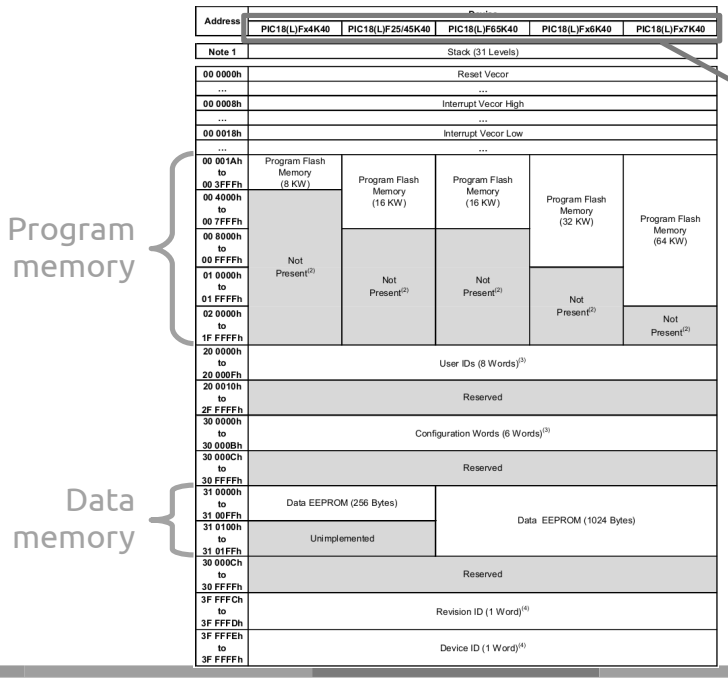
Finally you saw that the ALU performs simple operations. You should then avoid using advanced operators such as '/', '%', …

C-Type	custom typedef	Memory space	Values
char	int8	8 bits / 1 byte	-128 / 127
unsigned char	uint8	8 bits / 1 byte	0 / 255
short	int16	16 bits / 2 bytes	-32768 / +32767
unsigned short	uint16	16 bits / 2 bytes	0 / +65535
long	int32	32 bits / 4 bytes	-2G / +2G
unsigned long	uint32	32 bits / 4 bytes	0 / +4G
long long	int64	64 bits / 8 bytes	-9E / +9E
unsigned long long	uint64	64 bits / 8 bytes	0 / +18E
int		processor dependant	processor dependant
unsigned int		processor dependant	processor dependant
float		32 bits / 4 bytes (PIC/XC8)	
double		32 bits / 4 bytes (PIC/XC8)	

MEMORY

MEMORY

Program and data memory map



Size of program and data memories depends on the device

MEMORY

PIC18F4550 program memory map

Program Counter (Fetch stage)

Hardware stack, max 31 nested function calls (saving return addresses)

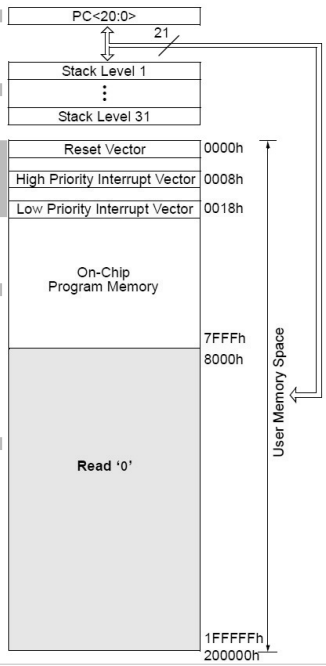
Relocatable table of interrupt vectors (redirections from IRQ to ISR)

Memory space for the embedded firmware (32 kB for the PIC18F4550 shown here)

Empty space depending on the device

Some PIC18 have up to 128 kB of program memory

Remaining space might be used for future evolutions



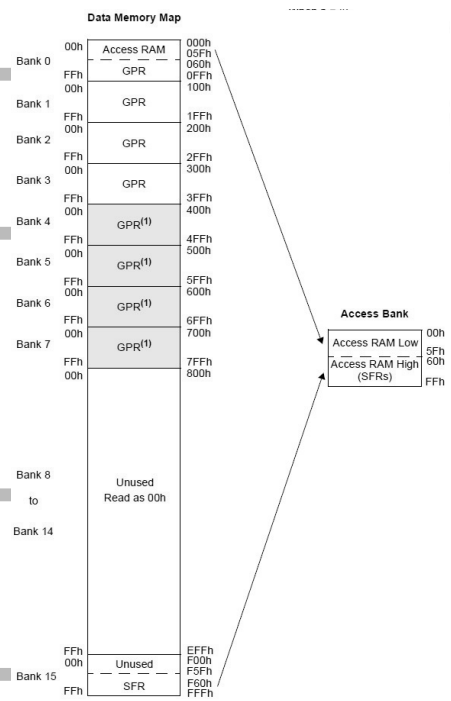
PIC18F4550 data memory map

Data memory segmented in 16 banks of 256 bytes
(see next page for selecting the working bank)

GPR (General Purpose Register file)
Registers that can be used to store any data,
available for use by all instructions.

2 kB data memory for this PIC18F4550
Up to 4kB for some other PIC18

SFR (Special Function Registers),
linked to CPU and peripherals registers



Data memory: Bank selection

The data memory is segmented in 16 banks of 256 bytes. This is a typical construction on 8-bit architectures.

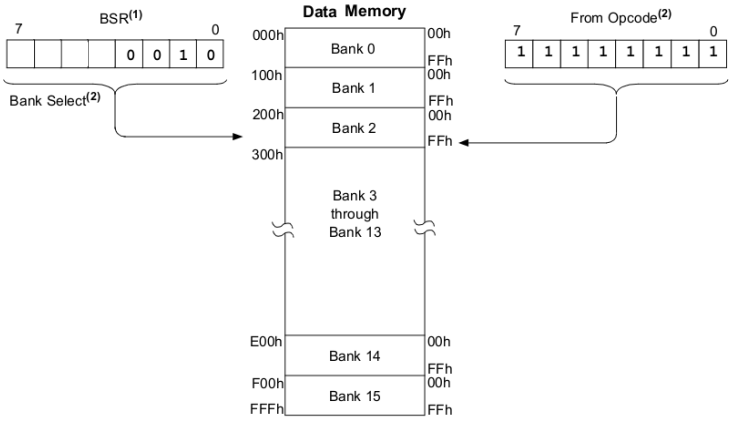
The working bank is selected by configuring four bits in the BSR (Bank Select Register).

Then instructions that use a file-register operand will access to the data with 8-bit direct addressing mode.

Bank select

PIC18 C language:
BSR = 0x02;

PIC18 assembly language:
```
MOVLW 0x02
MOVWF BSR
```



From the 16-bit opcode

PIC18 assembly language:
```
ADDWF f, d, a
```

16-bit opcode:
```
0010 01da ffff ffff
```

MEMORY

Data memory: Special Function Registers (SFR)

The SFR bank is a part of the RAM data memory where CPU and peripheral registers are mapped.

It means registers are physically outside of the memory but they are accessible with a memory address.

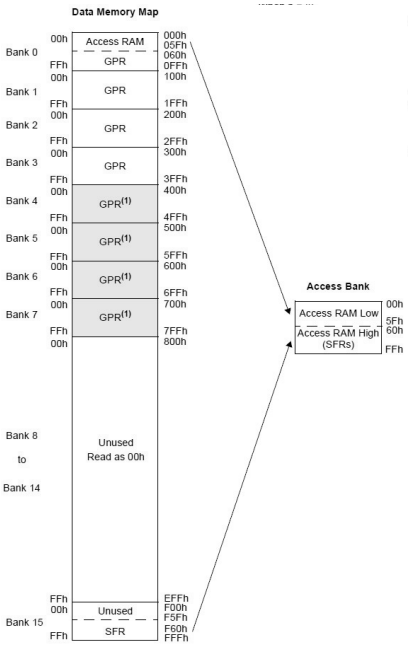
Core memory mapped registers

Peripheral specialised functions memory mapped registers

Address	Name	Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2(1)	FBFh	CCPR1H	F9Fh	IPR1	F7Fh	UEP15
FFEh	TOSH	FDEh	POSTINC2(1)	FBEh	CCPR1L	F9Eh	PIR1	F7Eh	UEP14
FFDh	TOSL	FDDh	POSTDEC2(1)	FBDh	CCP1CON	F9Dh	PIE1	F7Dh	UEP13
FFCh	STKPTR	FDCh	PREINC2(1)	FBCh	CCPR2H	F9Ch	—(2)	F7Ch	UEP12
FFBh	PCLATU	FDBh	PLUSW2(1)	FBBh	CCPR2L	F9Bh	OSCTUNE	F7Bh	UEP11
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	—(2)	F7Ah	UEP10
FF9h	PCL	FD9h	FSR2L	FB9h	—(2)	F99h	—(2)	F79h	UEP9
FF8h	TBLPTRU	FD8h	STATUS	FB8h	BAUDCON	F98h	—(2)	F78h	UEP8
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	ECCP1DEL	F97h	—(2)	F77h	UEP7
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	ECCP1AS	F96h	TRISE(3)	F76h	UEP6
FF5h	TABLAT	FD5h	T0CON	FB5h	CVRCON	F95h	TRISD(3)	F75h	UEP5
FF4h	PRODH	FD4h	—(2)	FB4h	CMCON	F94h	TRISC	F74h	UEP4
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB	F73h	UEP3
FF2h	INTCON	FD2h	HLVDCON	FB2h	TMR3L	F92h	TRISA	F72h	UEP2
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	—(2)	F71h	UEP1
FF0h	INTCON3	FD0h	RCON	FB0h	SPBRGH	F90h	—(2)	F70h	UEP0
FEFh	INDF0(1)	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—(2)	F6Fh	UCFG
FEEh	POSTINC0(1)	FCEh	TMR1L	FAEh	RCREG	F8Eh	—(2)	F6Eh	UADDR
FEDh	POSTDEC0(1)	FCDh	T1CON	FADh	TXREG	F8Dh	LATE(3)	F6Dh	UCON
FECh	PREINC0(1)	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD(3)	F6Ch	USTAT
FEBh	PLUSW0(1)	FCBh	PR2	FABh	RCSTA	F8Bh	LATC	F6Bh	UEIE
FEAh	FSR0H	FCAh	T2CON	FAAh	—(2)	F8Ah	LATB	F6Ah	UEIR
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA	F69h	UIE
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	—(2)	F68h	UIR
FE7h	INDF1(1)	FC7h	SSPSTAT	FA7h	EECON2(1)	F87h	—(2)	F67h	UFRMH
FE6h	POSTINC1(1)	FC6h	SSPCON1	FA6h	EECON1	F86h	—(2)	F66h	UFRML
FE5h	POSTDEC1(1)	FC5h	SSPCON2	FA5h	—(2)	F85h	—(2)	F65h	SPPCON(3)
FE4h	PREINC1(1)	FC4h	ADRESH	FA4h	—(2)	F84h	PORTE	F64h	SPPEPS(3)
FE3h	PLUSW1(1)	FC3h	ADRESL	FA3h	—(2)	F83h	PORTD(3)	F63h	SPPCFG(3)
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC	F62h	SPPDATA(3)
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB	F61h	—(2)
FE0h	BSR	FC0h	ADCON2	FA0h	PIE2	F80h	PORTA	F60h	—(2)

MEMORY

Data memory: Access bank



When executing an operation that uses direct addressing, the execution stage checks the <a> field (access bank):

```
ADDWF f, d, a
```

<a> = '0' or '1'

If <a> = '0' the CPU only sees the **access bank (top half of the bank #0 + SFR bank)**. The 4 most significant bits of the 12-bit data memory address come from the access bank register (0x0 or 0xF) → **Fast solution**.

If <a> = '1' the CPU can access to all the data memory. It uses the value of the **BSR (Bank Select Register)** to generate the 4 most significant bits of the 12-bit data memory address.

MEMORY

Data memory

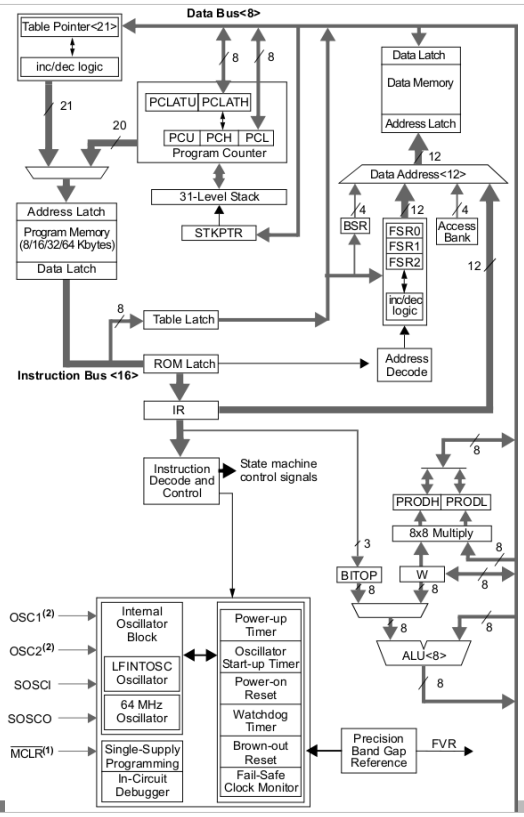
Find following items in this schematic

Data memory

BSR (Bank Select Register)

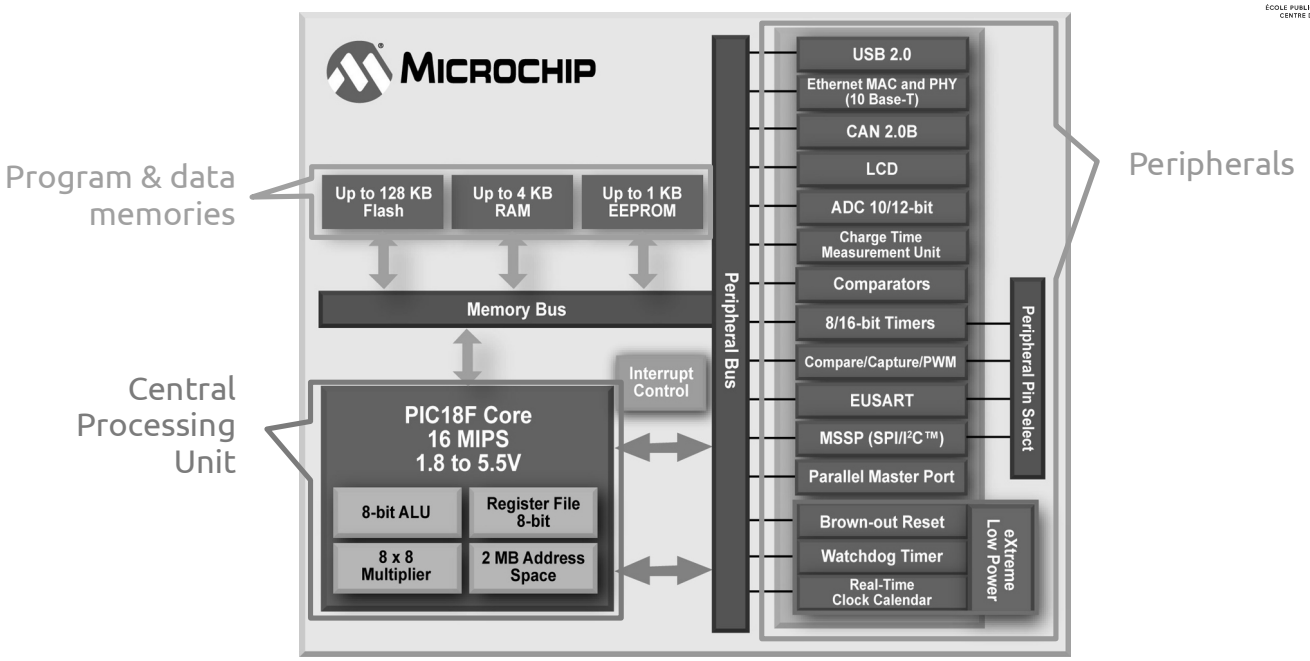
Access bank register

Note how they are placed relatively to each other and how the 12-bit data memory address is built.



MEMORY

PIC18 architecture



PERIPHERALS

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

PERIPHERALS

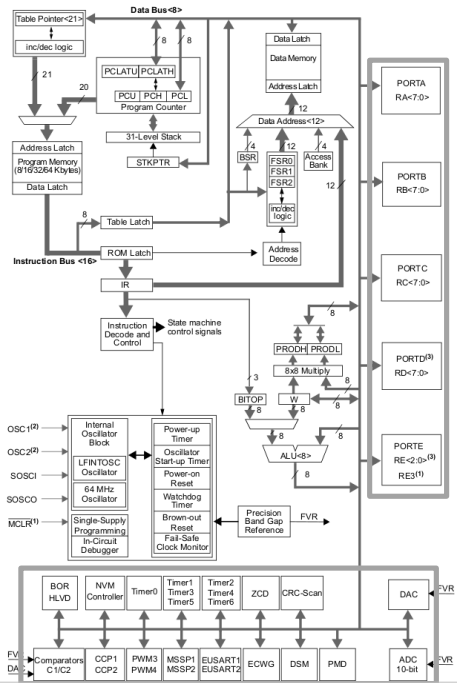
Definition

A peripheral is a hardware function that can be used to perform specific calculus and/or process some operation while letting the CPU performing something else.

For all MCUs, peripherals are physically connected to the data bus and their internal registers are mapped to the data memory.

From the program point of view, accessing to a register (read or write) is the same as accessing to a data memory cell.

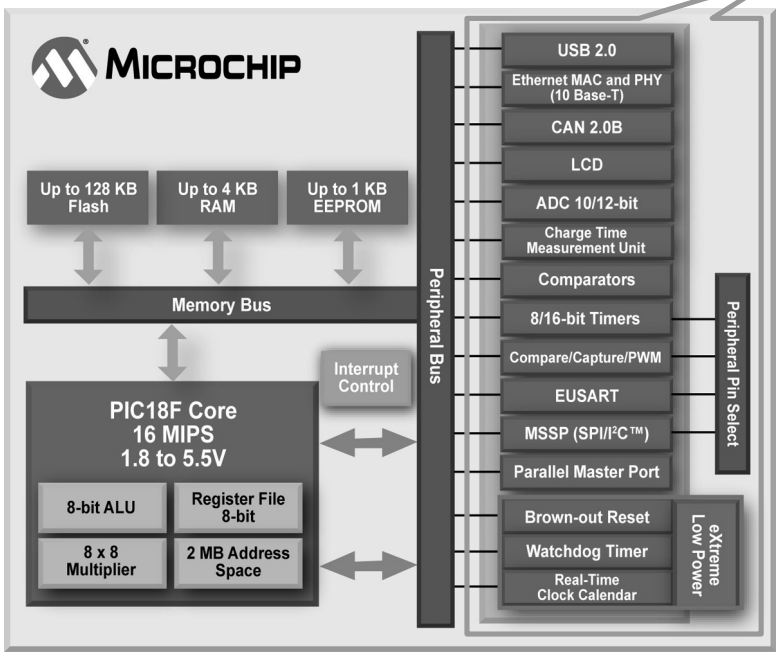
Example of PIC18F27/47K40 CPU and peripherals.



PERIPHERALS

Hardware functions

Peripherals,
guess what they do!



This lecture does not contain any material for using each peripheral. To know what is the role of a peripheral and how to configure it, the best material is the device datasheet.

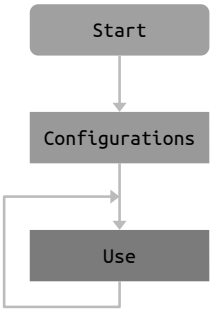
PERIPHERALS

Configuration, activation, use

When the processor starts (after power-on or reset), no peripheral function is configured nor activated.

The programmer must explicitly configure and activate hardware services that are needed for the application. Only then the peripherals can be used.

Most of peripherals have configuration registers that must be set once in addition to working registers that contains updated values.



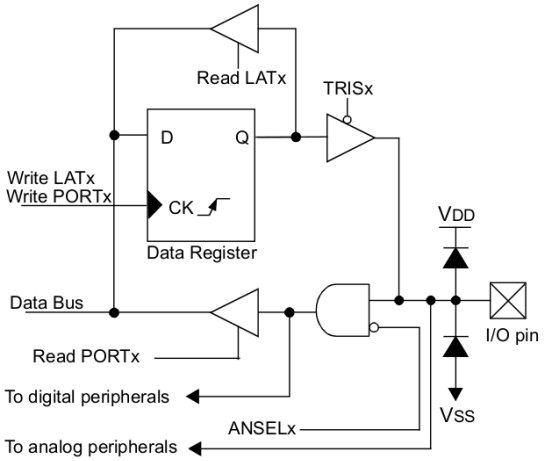

```
void main()
{

      // Peripherals configurations
      // and other initialisation
      …

      while(1) {
            // Main routine and peripherals use
            …
      }
}
```

A port is a group of 8 pins or GPIOs (General Purpose Input/Output).

They can be used as independent digital inputs and/or outputs. The PIC18F27K40 has 5 ports (port A to port E), which makes 40 independent GPIOs available.

Each port has eight registers to control the operation. These registers are:

- PORTx registers (reads the levels on the pins of the device)
- LATx registers (output latch)
- TRISx registers (data direction)
- ANSELx registers (analog select)
- WPUx registers (weak pull-up)
- INLVLx (input level control)
- SLRCONx registers (slew rate control)
- ODCONx registers (open-drain control)



Example of **GPIO (General Purpose Input/Output)** on the Curiosity HPC board.

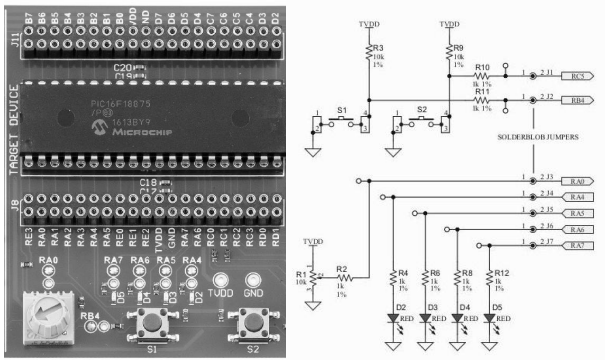
First the RA4 pin is configured as an output, then the output value is set to 'high'. This will turn on the LED D2 on this pin.

PIC18 C program

```
//Set RA4 as an output
TRISA = 0xEF;

//Set RA4 to high level
LATD = 0x10;
```

PIC18 assembly program

```
;Set RA4 as an output
MOVLW    0xEF
MOVWF    TRISA

;Set RA4 to high level
MOVLW    0x10
MOVWF    LATA
```

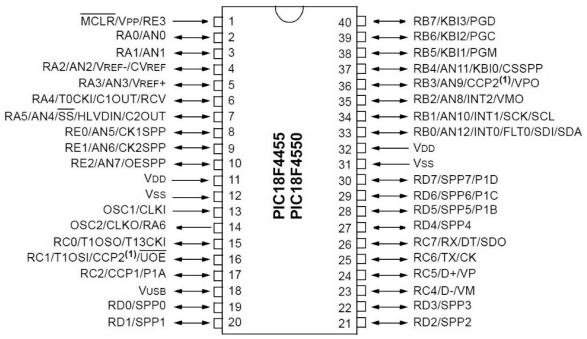


RA4 *= pin 4 of port A (also bit 4 of registers associated to port A).*
TRISx *= register that contains the direction of the 8 pins of the port x ('0' = Output, '1' = Input).*
LATx *= register that contains the output value for pins configured as outputs in port x.*

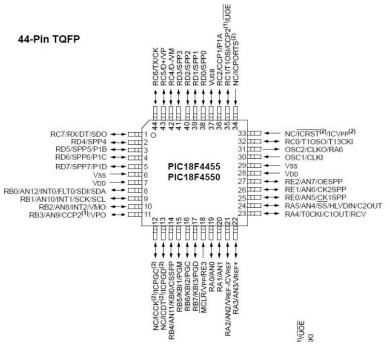
MCUs usually have more peripherals than they can really handle.

In fact, many peripherals are connected to the same pins. This implies that some peripherals cannot be used at the very same time.

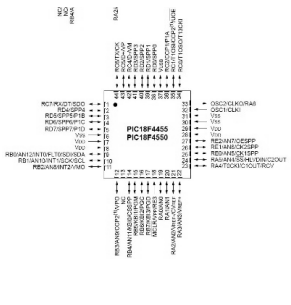
DIP package



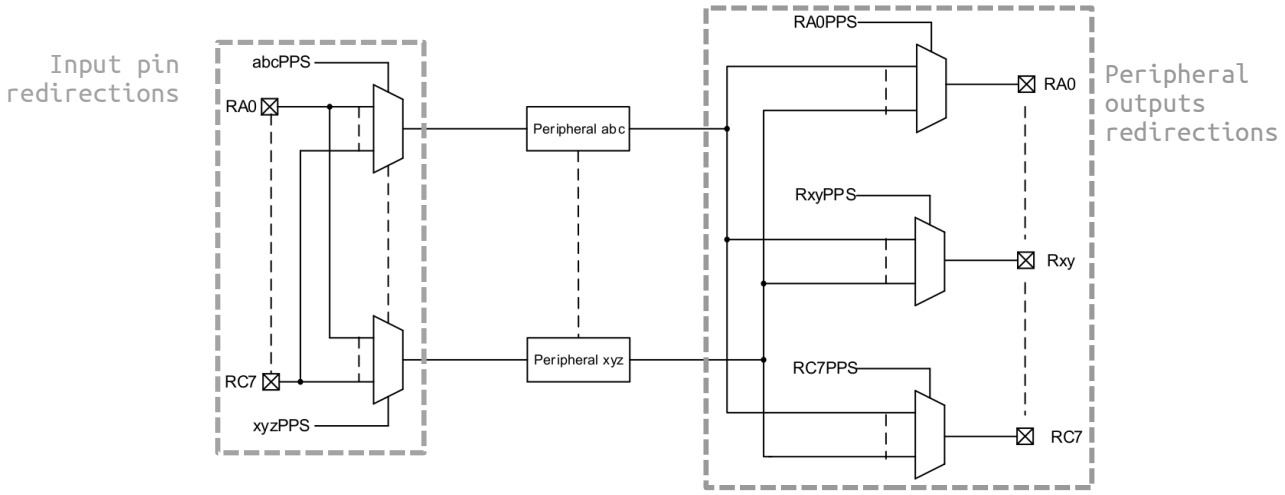
TQFP package



QFN package



As many input and output pins can be used by several peripherals, connections between pins and peripherals must be set using the Peripheral Pin Select (PPS) module.



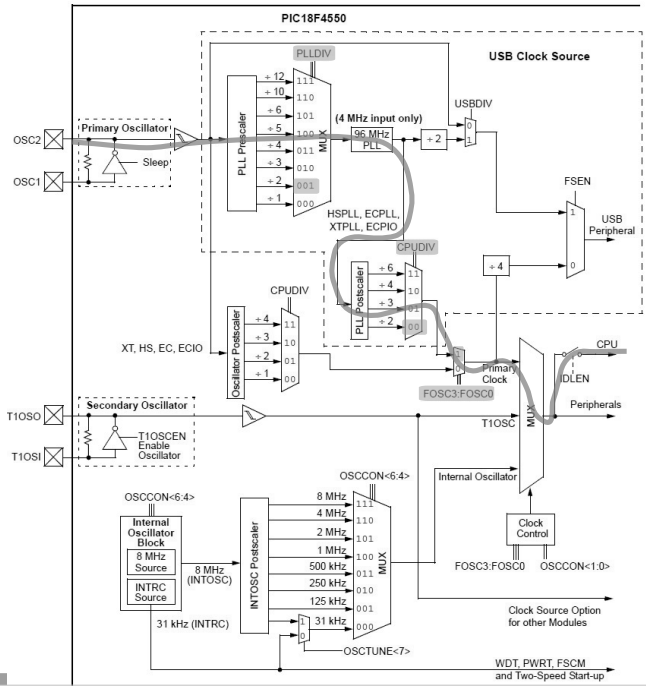
Like all other MCUs, the PIC18 family offers configuration and system hardening services.

"Hardening" an application consists in making it less sensitive to its environment (power supply fluctuations, …) or even handle unexpected situations (watchdog, …).

Any program on PIC18 must start with this configuration:

```
/* CPU specific features configuration */
#pragma config PLLDIV=2, CPUDIV=OSC1_PLL2, FOSC=HSPLL_HS
#pragma config BOR=OFF, WDT=OFF, MCLRE=ON, LVP = OFF
```


External clock
(e.g. 8 MHz cristal)

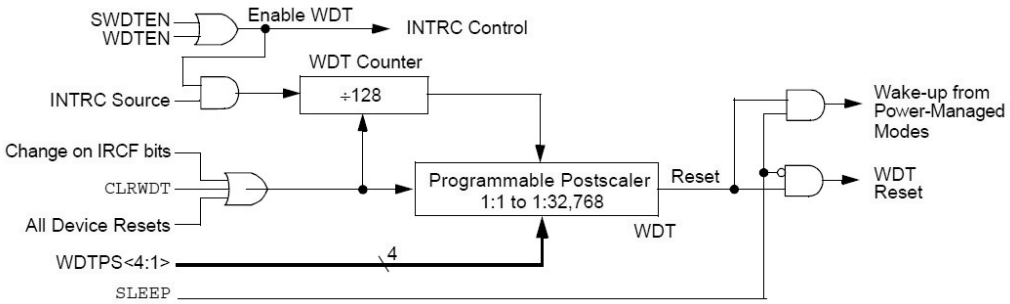



```
#pragma config
PLLDIV=2,
CPUDIV=OSC1_PLL2,
FOSC=HSPLL_HS
```

The watchdog is an application agent.

In normal circumstances, the application resets the watchdog timer at regular intervals.

When the application remains stuck for a long time, the watchdog timer is not reset and will eventually overflow. The watchdog will force the application to reboot by resetting the CPU.



INTERRUPTS

Event

Interrupt Flag (IF)

Interrupt Request (IRQ)

Interrupt Service Routine (ISR)

Interrupt flag

Once a peripheral has been configured, it becomes autonomous.

When its job is done (counting, converting, …) or **when a special event occurs** (message received, switch pressed, …), **the peripheral will raise a Interrupt Flag (IF)**.

A interrupt flag is a bit in a register, each flag corresponding to a precise event.

Peripheral Interrupt Request (Flag) Register 0

Bit	7	6	5	4	3	2	1	0
			TMR0IF	IOCIF		INT2IF	INT1IF	INT0IF
Access			R/W	R		R/W	R/W	R/W
Reset			0	0		0	0	0

Bit 5 – TMR0IF Timer0 Interrupt Flag bit(1)

Value	Description
1	TMR0 register has overflowed (must be cleared by software)
0	TMR0 register has not overflowed

Bit 4 – IOCIF Interrupt-on-Change Flag bit(1,2)

Value	Description
1	IOC event has occurred (must be cleared by software)
0	IOC event has not occurred

Bits 0, 1, 2 – INTxIF External Interrupt 'x' Flag bit(1,3)

Value	Description
1	External Interrupt 'x' has occurred
0	External Interrupt 'x' has not occurred

Interrupt Service Routine (ISR)

However some events require immediate attention. Plus, using interrupt flags the same way as classical variables (e.g. testing them in "`if`" statements) is inefficient.

That is why an **interrupt mechanism** has been designed. This is the hardware way of stopping the normal execution flow of the CPU. It will then switch to the execution of a function dedicated to the event: the **ISR (Interrupt Service Routine)**.

```
// Doing the classical way

main() {

        // do something
        ...
        ...
        ...
        ...
        while(1){
                if( event_A )
                ...
                if( event_B )
                ...
                if( event_C )
        }
}
```

Event C occurs here

Event C is processed here

```
// Using interrupt mechanism

void event_A_ISR()
{ ... }

void event_B_ISR()
{ ... }

void event_C_ISR()
{ ... }

main() {
        // do something
        ...
        ...
        ...
}
```

Event C occurs here

Immediate processing

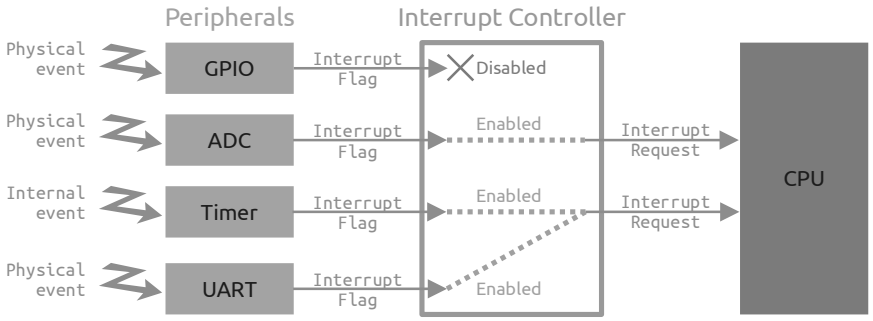
Main program execution is paused

Interrupt Request (IRQ)

All interrupts are disabled by default: it is the programmer work to decide and explicitly tell which event is worthy enough to lead to an interrupt.

To do so, the programmer must configure the **Interrupt controller**.

This circuit will turn selected interrupt flags into **Interrupt Requests (IRQ)**, that will cause the CPU to stop its current work.



Interrupt flag
Bit in a register that ***indicates*** that an event has occurred.

Interrupt request
Physical wire connected to the CPU that ***makes it switch to another function***.

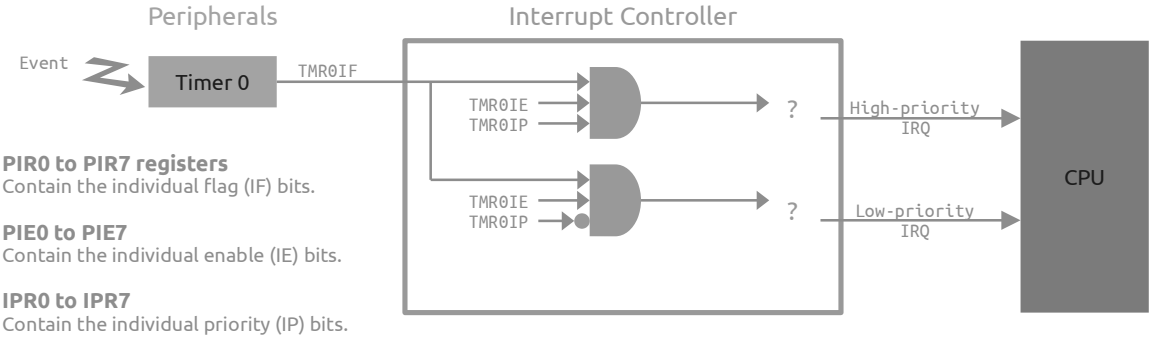
Notes
Many interrupt flags can be linked onto the same IRQ signal.
A specific IRQ corresponds to a specific ISR.

Interrupt controller

The **interrupt controller** consists of AND and OR gates. This circuitry independently let or prevent interrupt flags becoming interrupt requests.

To do so, one must configure the registers of the interrupt controller. Like the PIC18, most MCUs need to configure three bits for each interrupt:

the Interrupt Flag (IF) bit, the Interrupt Enable (IE) bit, the Interrupt Priority (IP) bit.



PIR0 to PIR7 registers
Contain the individual flag (IF) bits.

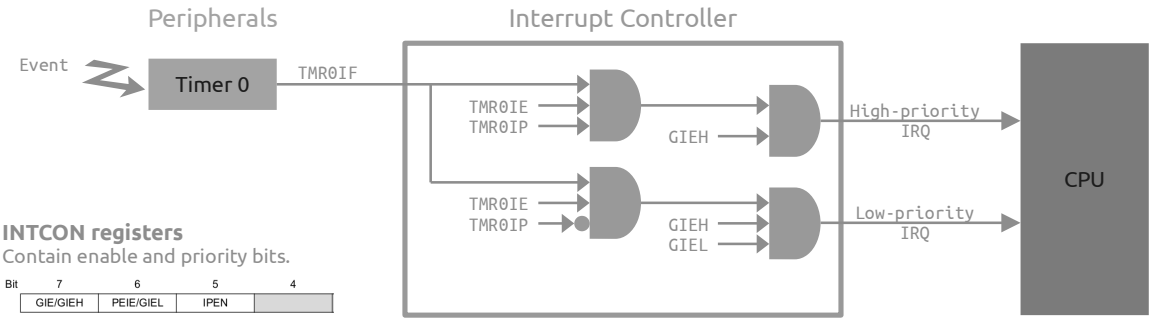
PIE0 to PIE7
Contain the individual enable (IE) bits.

IPR0 to IPR7
Contain the individual priority (IP) bits.

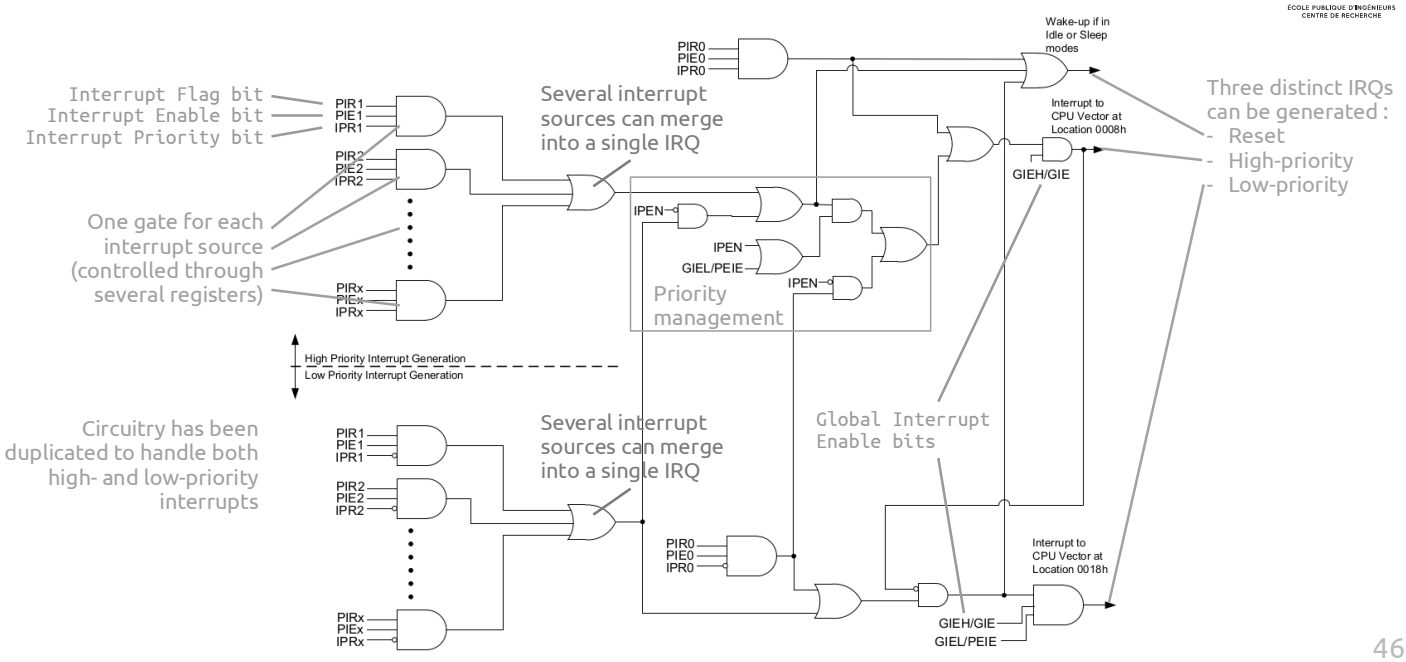
Interrupt controller

After configuring the interrupt controller for every single interrupt source, one last parameter should allow the CPU to see IRQ signals.

Most MCUs have a Global Interrupt Enable (GIE) bit to do so. The PIC18 has two of them: GIEH and GIEL for respectively high- and low-priority interrupts.

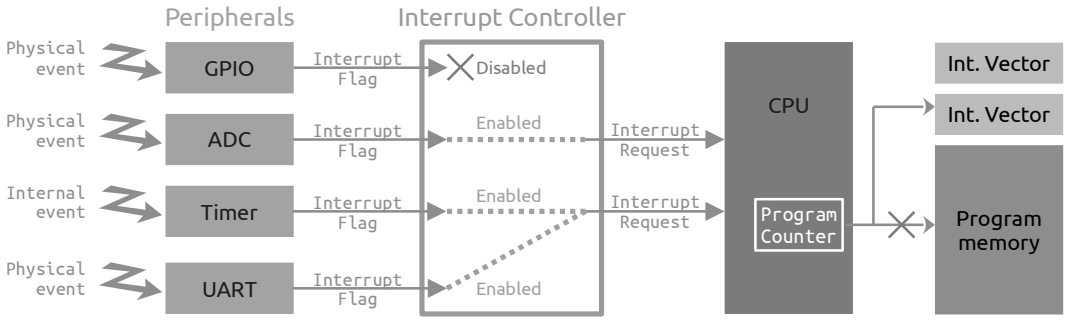


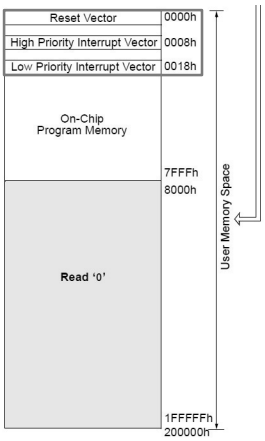
Interrupt controller: PIC18F27/47K40 interrupt controller logic



Interrupt Service Routines (ISR) are functions called by the **Interrupt vectors**, which are specific parts of the Flash memory.

When the CPU sees an **Interrupt Request (IRQ)**, it pauses its main program and branches automatically to the corresponding interrupt vector, causing the execution of the Interrupt Service Routine.





ISRs are not proper functions: they should not be called from the main program.

They are automatically called whenever the CPU sees the IRQ signals (this is called "event-driven programming", fr: *programmation évènementielle*)

As ISRs are called at unpredictable moments, it is not possible to pass arguments to the ISR functions.

In order to exchange information between the main program and the ISRs, global variables can be used.

But remember global variables are shared resources and be used very carefully!

Interrupt Service Routine (ISR)

Here is an example of how writing an ISR using the Microchip XC8 toolchain.

```
 /*  ISR - high level Interrupt Service Routine  */
void interrupt high_priority high_isr(void) {

     if( PIR0bits.TMR0IF ) {
          PIR0bits.TMR0IF = 0;
          ...
     }

     if( PIR1bits.ADIF ) {
          PIR1bits.ADIF = 0;
          ...
     }
}


/* program entry point */
void main(void) {

     timer0_init();
     interrupt_enable();

     while(1) {
          /* user application scheduler */
     }
}
```

- ISR for all high-priority interrupt sources
- Check if Timer 0 is the source of this interrupt
- Clear the Timer 0 interrupt flag, and proceed to what it should do
- Check if ADC is the source of this interrupt
- Clear the ADC interrupt flag, then proceed to what it should do
- Configure interrupt for Timer 0
- Enable interrupt for the CPU
- Main routine (main application function)

Interrupt vector

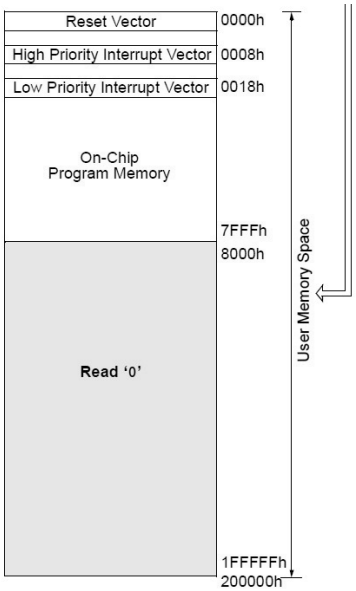
Interrupt vectors are very small areas in the Flash memory.

→ 16 bytes for the high-priority interrupt vector

→ 2 bytes for the low-priority interrupt vector

In fact, those areas are too small to contain the ISR but they are large enough to contain `CALL` or `GOTO` instructions, which will actually call the ISR (see next page).

Technically ISR are stored in the program memory, but they should be accessed only through interrupt vectors.



Context switch

As an ISR can be called at anytime, it will break the **context** that the main program is using (values of W-reg, STATUS, BSR for the PIC18). All registers must be saved in order to return to prior values when the ISR has ended.

The PIC18 toolchain will usually implement a hardware context backup for the high-priority ISR and a software context backup for the low-priority ISR.

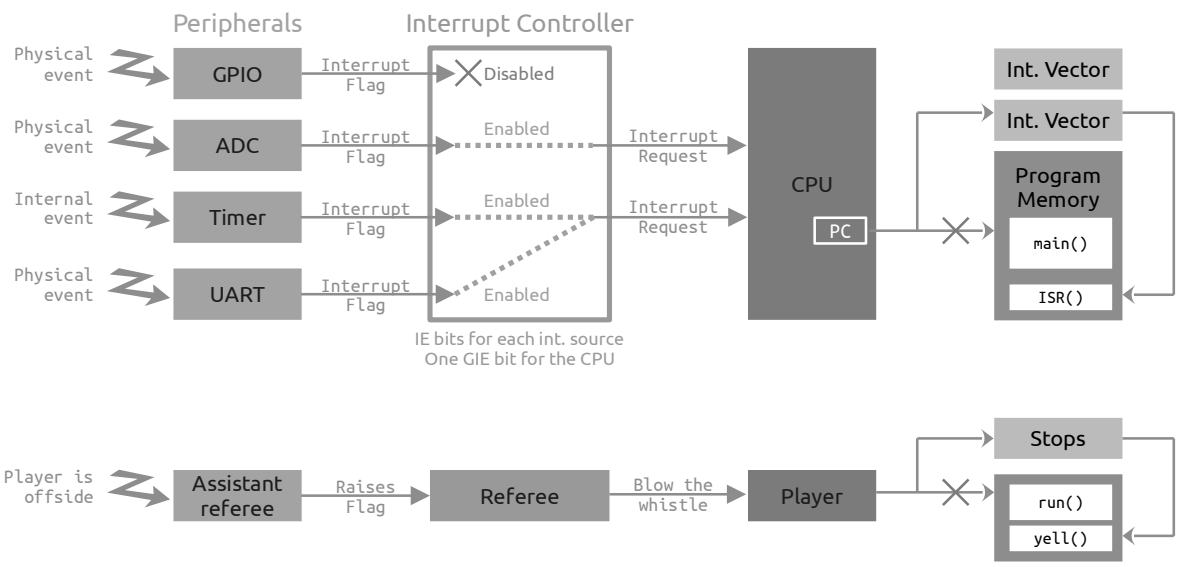
Hardware context backup
(CPU shadow registers)

```
high_vector:
      CALL  high_isr, 1

high_isr:
      ; user program - ISR processing
      RETFIE      1
```

Hardware context backup
(Flash memory)

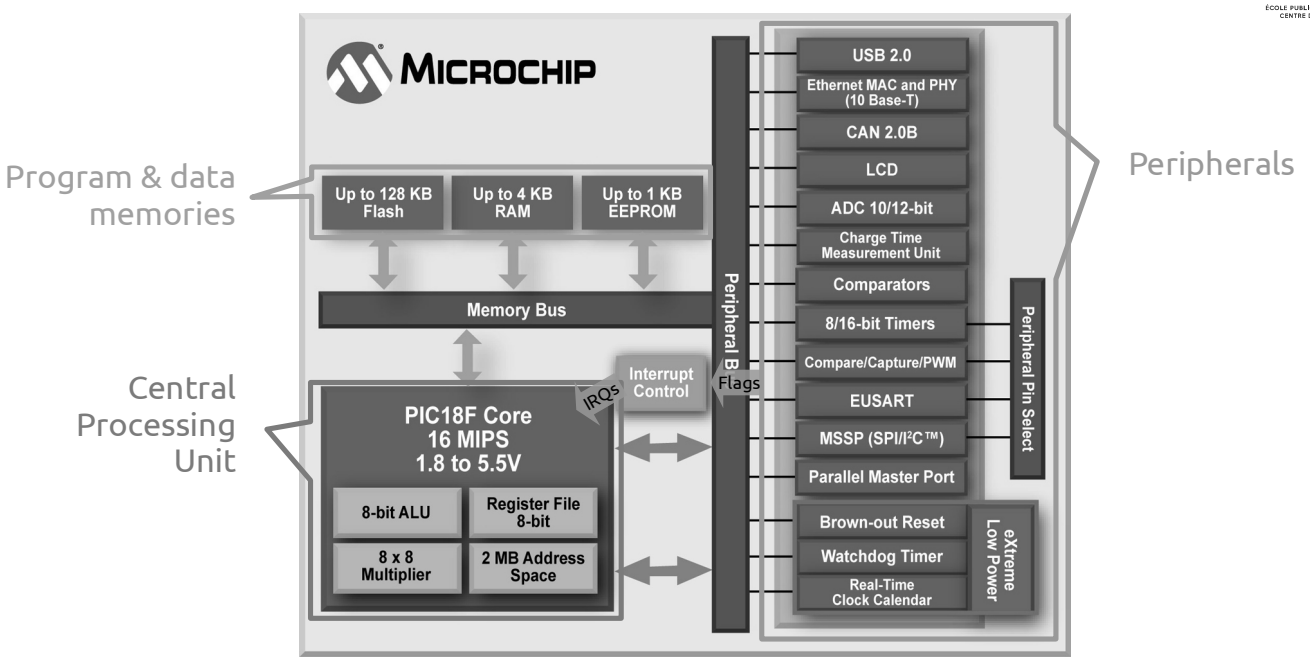
```
low_vector:
      GOTO    low_isr

low_isr:
      MOVWF   wreg_tmp
      MOVFF   STATUS,     status_tmp
      MOVFF   BSR,        bsr_tmp
      ; user program - ISR processing
      MOVFF   bsr_tmp,    BSR
      MOVFF   status_tmp, STATUS
      MOVF    wreg_tmp,   W
      RETFIE
```


Summary

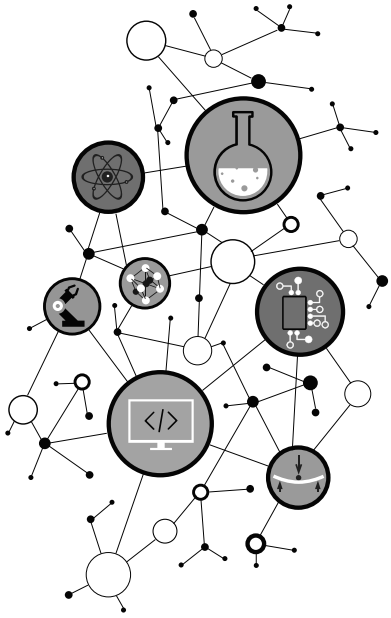


INTERRUPTS

PIC18 architecture



CONTACT



Dimitri Boudier – PRAG ENSICAEN

dimitri.boudier@ensicaen.fr

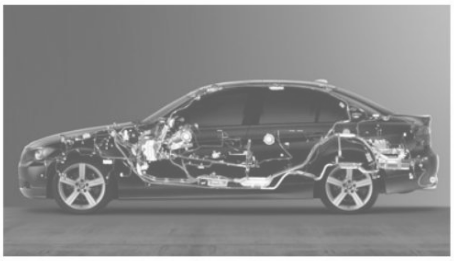
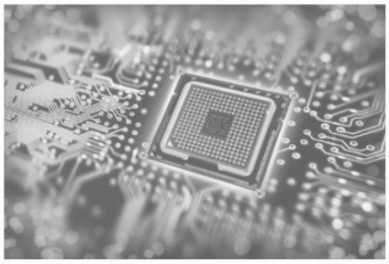
With the precious help of:

- Hugo Descoubes (PRAG ENSICAEN)
- Bogdan Cretu (MCF ENSICAEN)

ASSEMBLEUR PIC18

SYSTÈMES EMBARQUES

ASSEMBLEUR PIC18

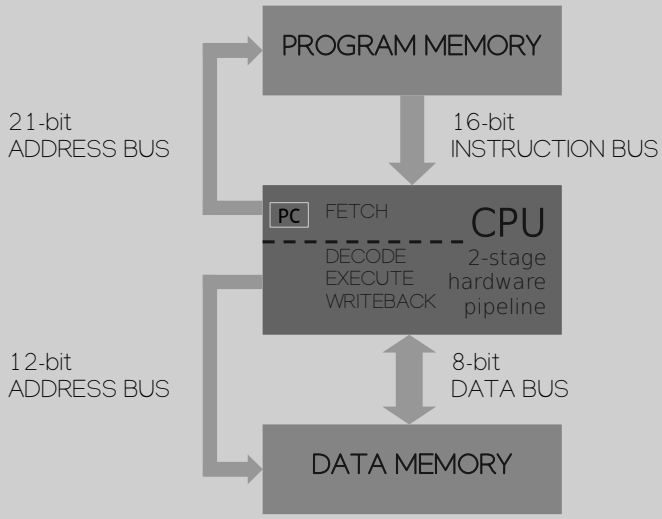



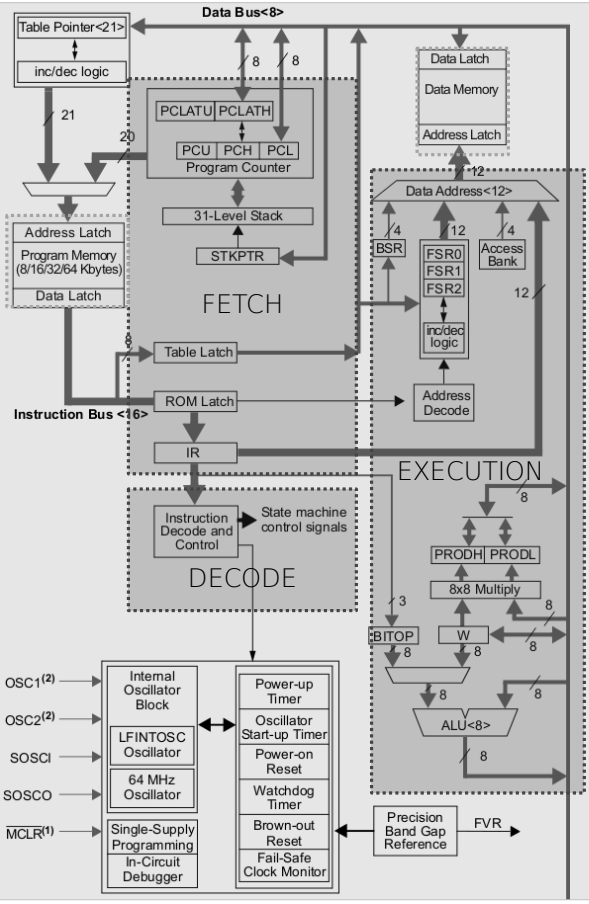





neku - hugo descoubes – enseignant Systèmes Embarqués – ENSICAEN - France
GNU\Linux Ubuntu 20.04 LTS – LibreOffice 6.4.6.2 – 2023

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	2	1100	ffff	ffff	ffff	None	
		f_d (destination) 2nd word		1111	ffff	ffff	ffff		
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	$\overline{TO}$, $\overline{PD}$	

SYSTÈMES EMBARQUES
ISA – Instruction Set Architecture ou jeu d'instructions PIC18

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	2	1110	1110	00ff	kkkk	None	
		to FSR(f) 1st word		1111	0000	kkkk	kkkk		
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

SYSTÈMES EMBARQUES
Format binaire ou opcode (code opératoire) des instructions

Byte-oriented file register operations

15 – 10	9	8	7 – 0
OPCODE	d	a	f (FILE #)

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)

15 – 12	11 – 0
OPCODE	f (Source FILE #)
1111	f (Destination FILE #)

f = 12-bit file register address

Example Instruction: `MOVFF MYREG1, MYREG2`

Bit-oriented file register operations

15 – 12	11 – 9	8	7 – 0
OPCODE	b (BIT #)	a	f (FILE #)

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

Example Instruction: `BSF MYREG, bit, B`

Literal operations

15 – 8	7 – 0
OPCODE	k (literal)

k = 8-bit immediate value

`MOVLW 7Fh`

Control operations

CALL, GOTO and Branch operations

15 – 8	7 – 0
OPCODE	n<7:0> (literal)

`GOTO Label`

15 – 12	11 – 0
1111	n<19:8> (literal)

n = 20-bit immediate value

15 – 8	7	0
OPCODE	S	n<7:0> (literal)

`CALL MYFUNC`

15 – 12	11 – 0
1111	n<19:8> (literal)

S = Fast bit

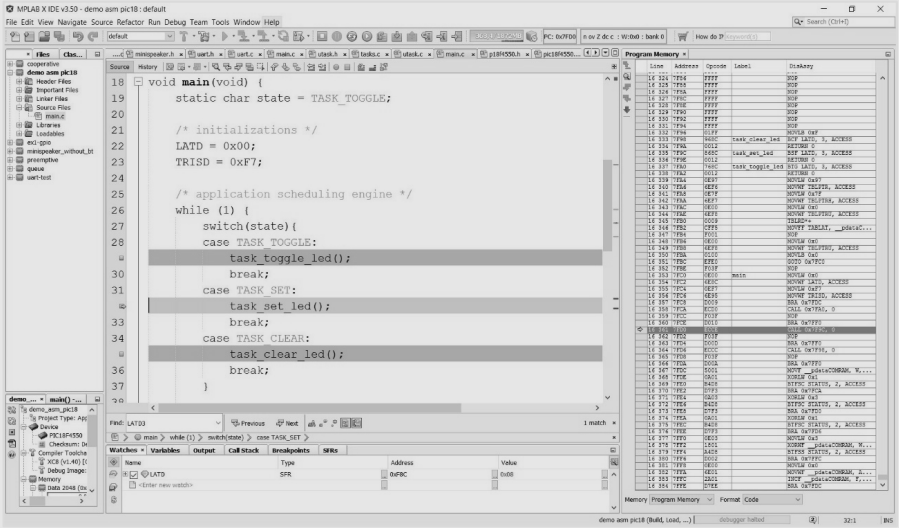
15 – 11	10 – 0
OPCODE	n<10:0> (literal)

`BRA MYFUNC`

15 – 8	7 – 0
OPCODE	n<7:0> (literal)

`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)



Programme à traduire

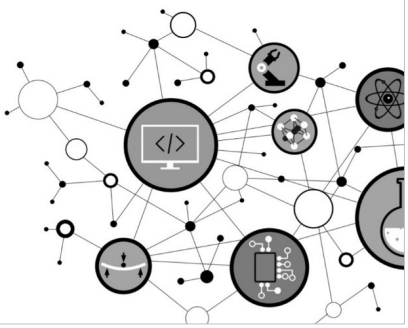
Page 1

```
/* 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;
```

Page 2

```
          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

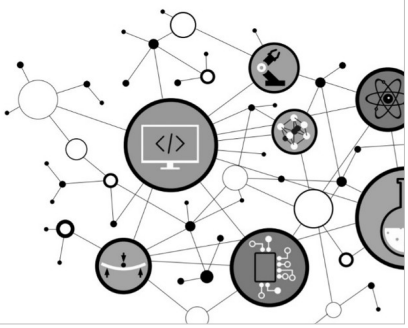


De façon générale, les squelettes des applications seront développés en C. Ceci facilite la lisibilité, l'édition et la maintenance du code. L'assembleur sera en générale utilisé afin de développer des procédures ou fonctions spécifiques, le plus souvent dans une optique d'optimisation (empreinte mémoire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

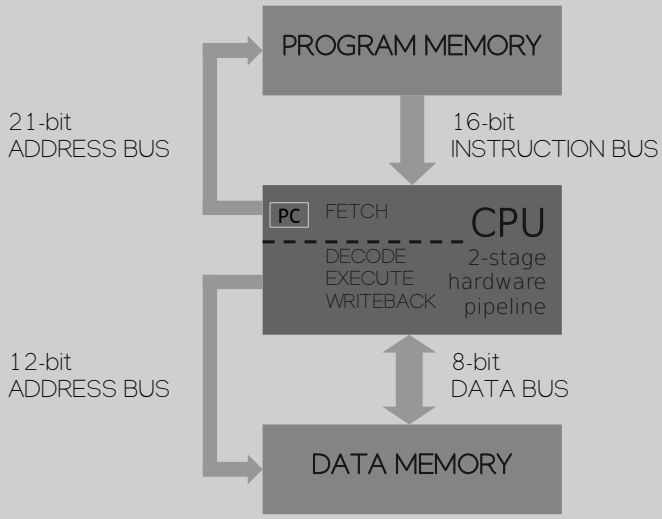
```
void task_toggle_led_D2 (void) {
     static char toggle = 0;

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

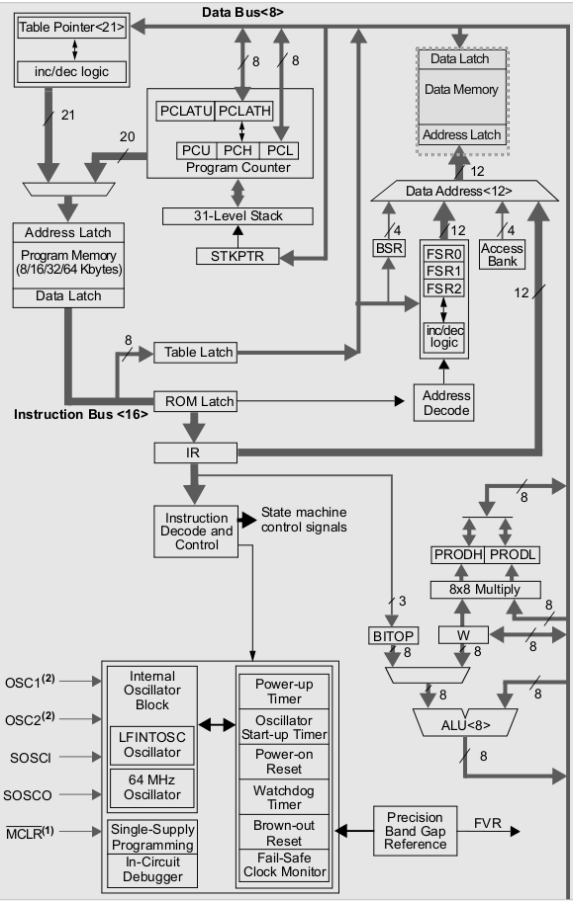
```
void task_toggle_led_D2 (void) {
    #asm
        BTG  LATA, 4
    #endasm
}
```

- 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

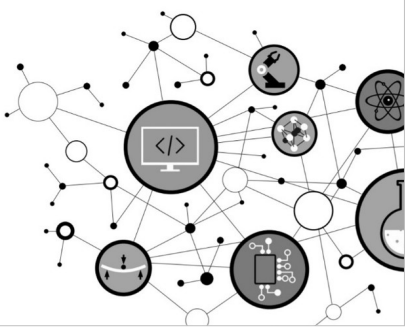
XC8

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

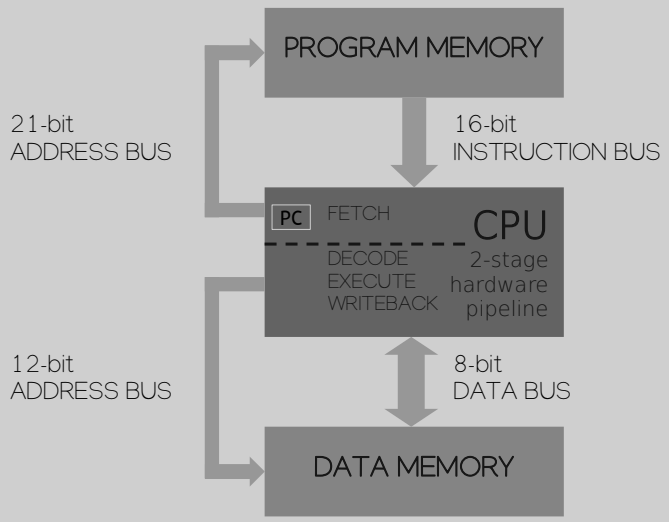
MPASMWIN

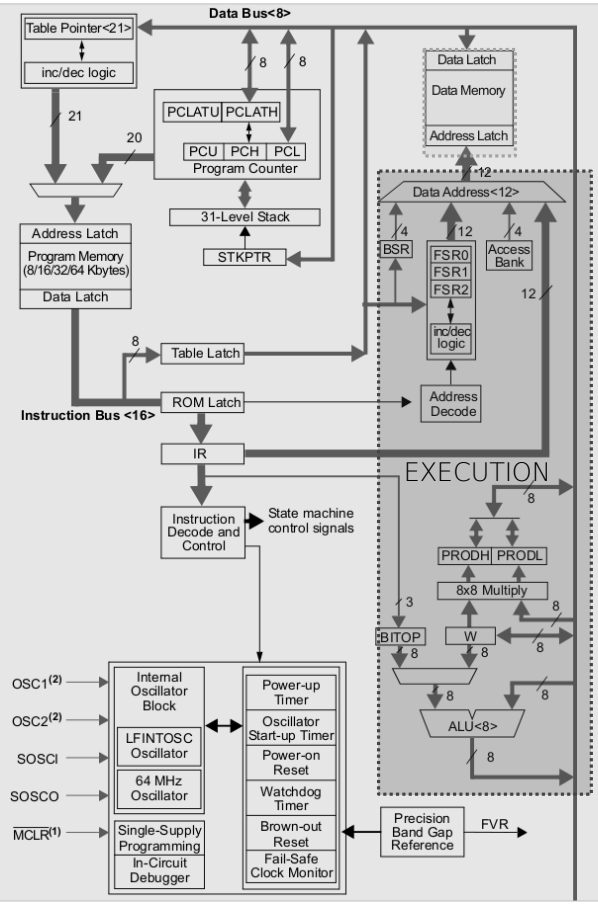
```
; reserve 1 byte in access bank
    idata_acs
state   db      0
```


- 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





Adressage immédiat

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

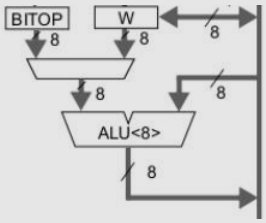
SYSTÈMES EMBARQUES
Adressage immédiat – toutes les instructions !

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 1111	1110 0000	00ff kkkk	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

15 – 8	7 – 0
OPCODE	k (literal)

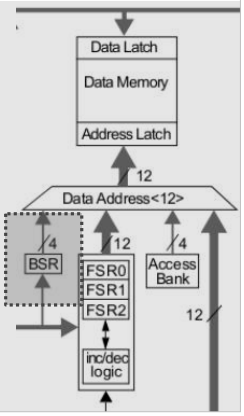
k = 8-bit immediate value



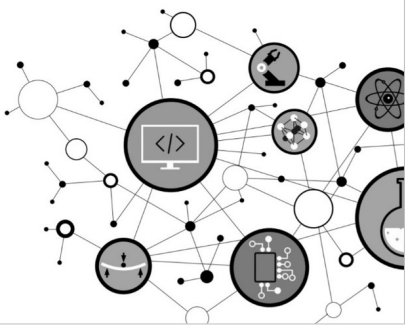
SYSTÈMES EMBARQUES
Adressage immédiat – sélection d'une banque en mémoire

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

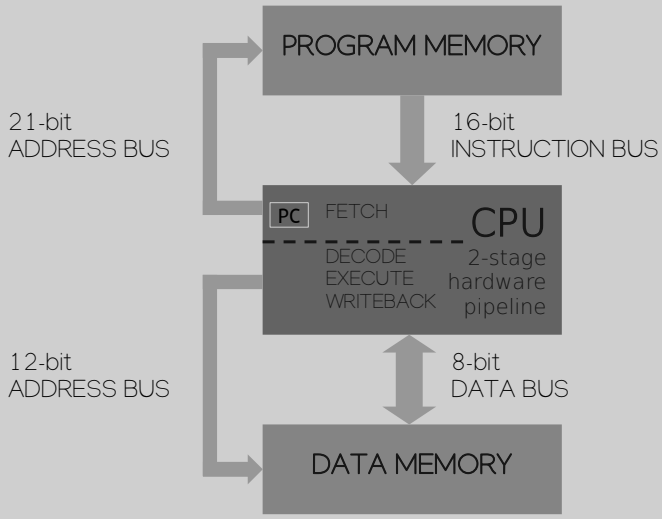
```
; access bank (bank0 or bank15)
; bank0   : 0x000-0x05F (GPR)
; bank15 : 0xF60-0xFFF  (SFR)
ADDWF        <8bits_relative_address_data_memory>, 0, 0

Autre solution :

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

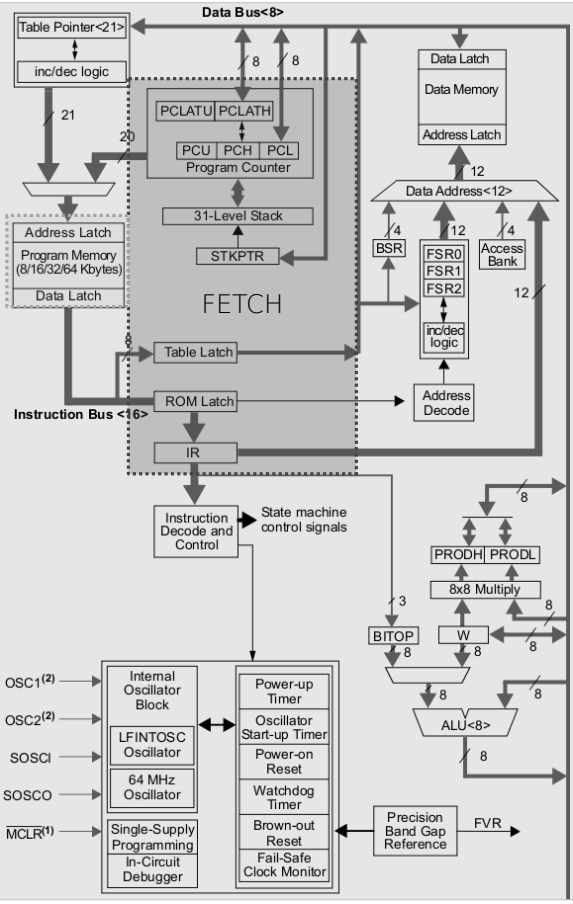


- Insertion ASM dans C
- Allocation statique de variable
- Adressage immédiat
- **Instruction de contrôle**
- Instructions orientées octet
- Instructions orientées bit
- Solution ASM
- Divers



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	PIC18 assembler Program

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

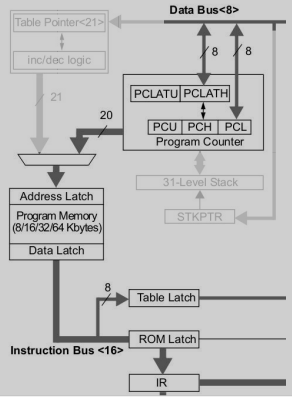
GOTO instruction Datasheet

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



Instructions de contrôle – saut absolu et relatif

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
```


Instructions de contrôle – saut conditionnel

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
```

SYSTÈMES EMBARQUES
Registre STATUS ou registre d'état de l'ALU

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

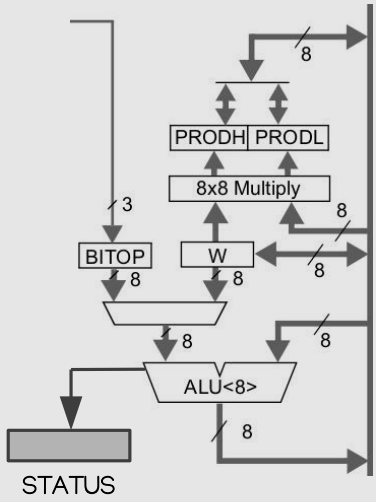
Bit 4 – N Negative bit
Used for signed arithmetic (2's complement); indicates if the result is negative, (ALU MSb = `1`).
Reset States: POR/BOR = 0
All Other Resets = u

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

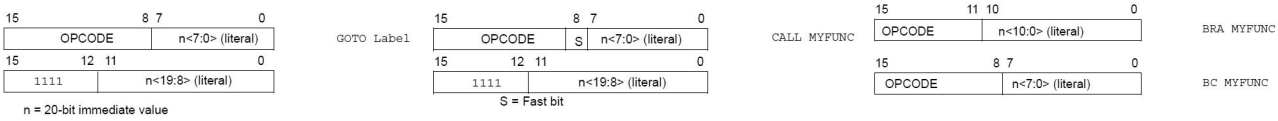
Bit 2 – Z Zero bit
Reset States: POR/BOR = 0



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

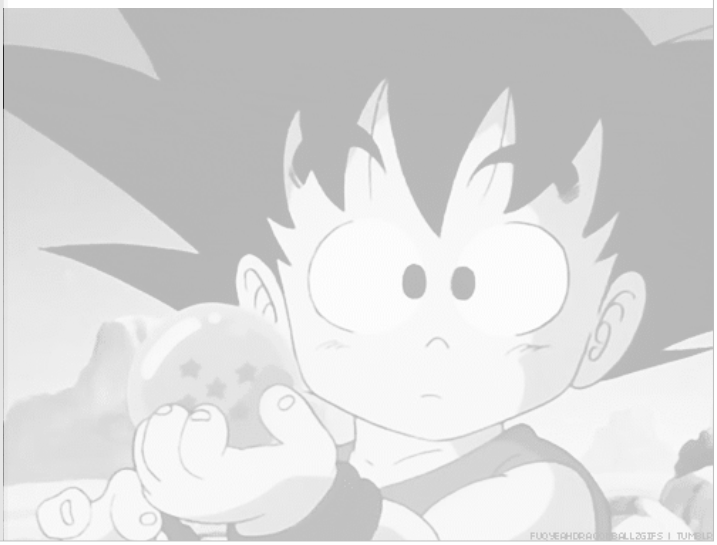
SYSTÈMES EMBARQUES
Instructions de contrôle – ISA

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 !?



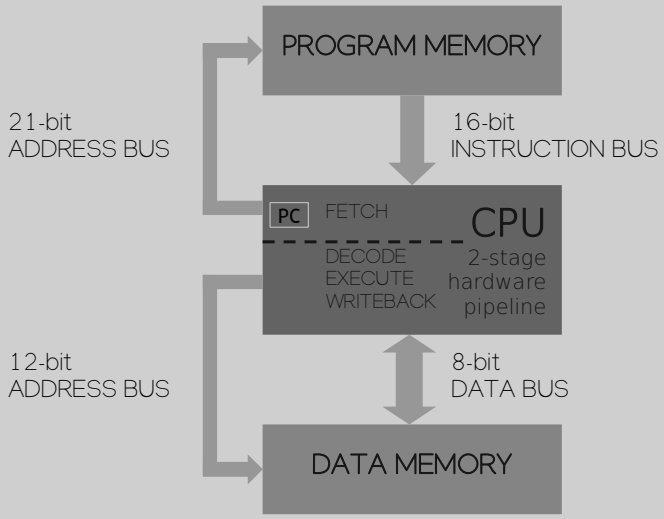

```
switch(state){
case TASK_TOGGLE:
    task_toggle_led_D2();
    break;
case TASK_SET:
    task_set_led_D2();
    break;
case TASK_CLEAR:
    task_clear_led_D2();
    break;
}
```

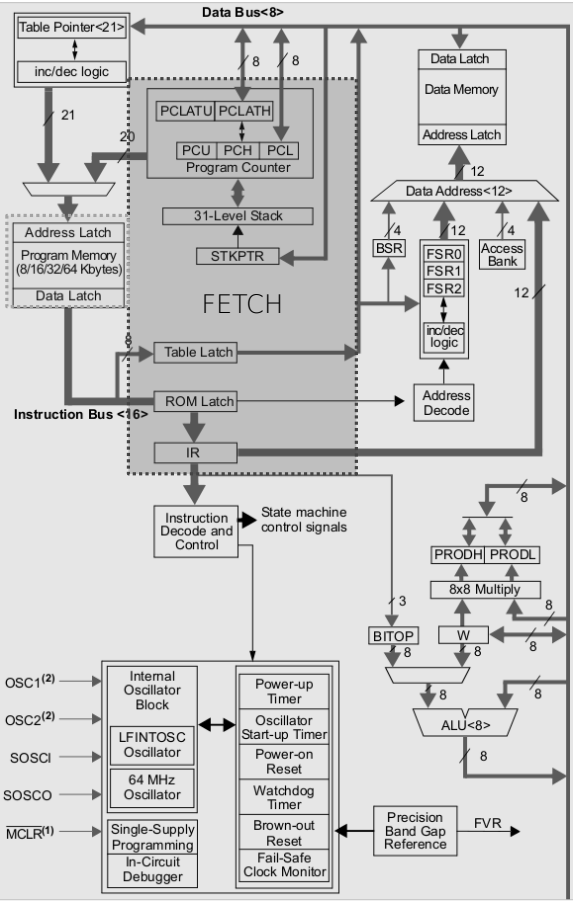
Mnemonic, Operands		Description
CONTROL OPERATIONS		
BC	n	Branch if Carry
BN	n	Branch if Negative
BNC	n	Branch if Not Carry
BNN	n	Branch if Not Negative
BNOV	n	Branch if Not Overflow
BNZ	n	Branch if Not Zero
BOV	n	Branch if Overflow
BRA	n	Branch Unconditionally
BZ	n	Branch if Zero
CALL	n, s	Call subroutine 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:    …
```

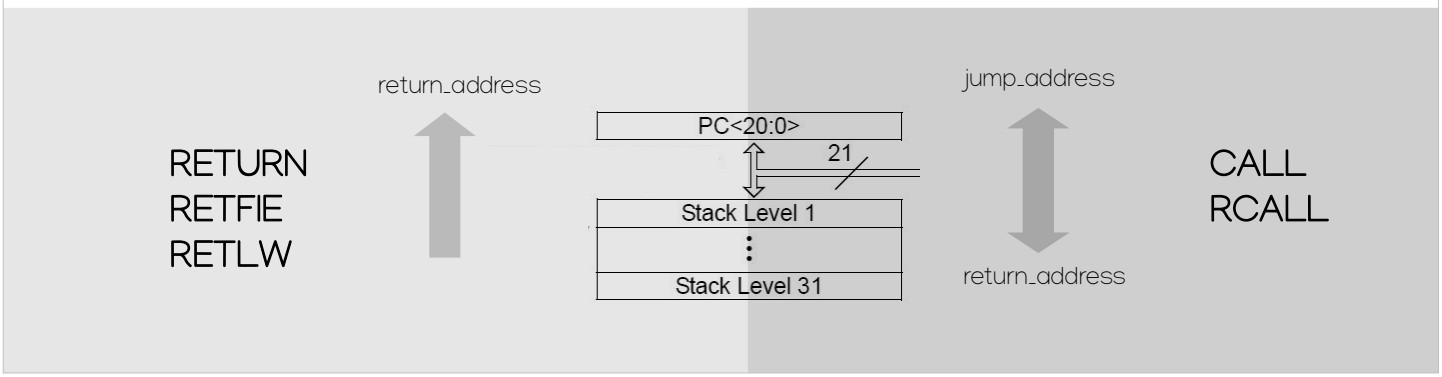
Architecture processeur MCU PIC18





Instructions de contrôle – Appel de fonction

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



Instructions de contrôle – Sauvegarde de contexte matérielle

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.

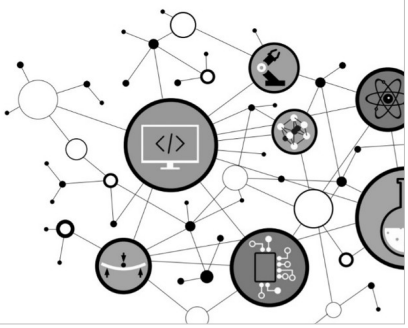
Then the 20-bit value 'k' is loaded into PC<20:1>. `CALL` is a two-cycle instruction.

Words:	2
Cycles:	2

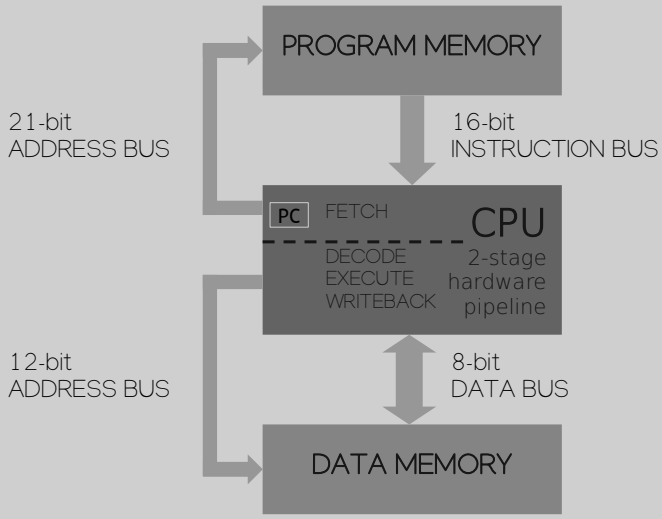
Top Of Stack pour l'adresse de retour

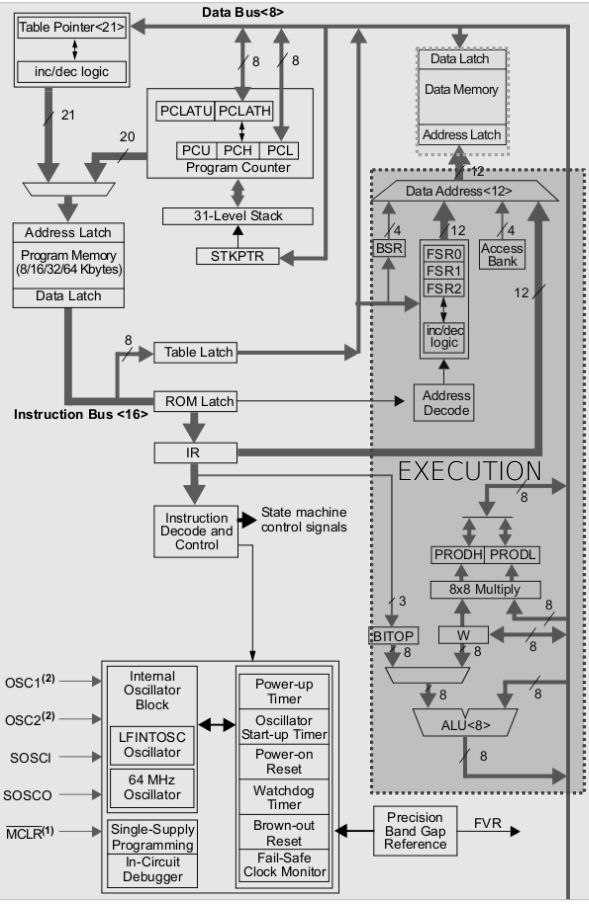
Le registres dont le nom est suffixé par S (Shadows Registers) sont cachés des développeurs et utilisés durant les sauvegardes de contexte matérielles avec l'usage d'interruption. Cette méthode accélère le mécanise de commutation

- 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





Instructions orientées octet

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;
}
```

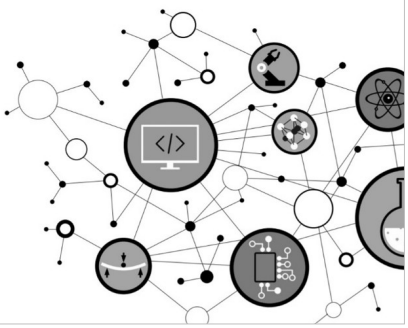
PIC18 assembler program

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


Instructions orientées octet - ISA

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 1111	ffff ffff	ffff 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`

SYSTÈMES EMBARQUES
Instructions orientées bits - ISA

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

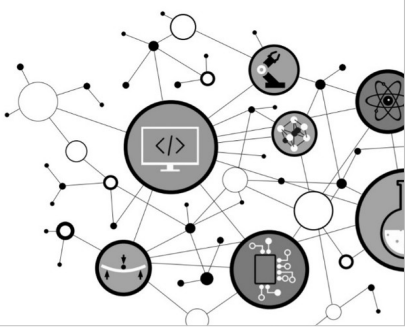
15 – 12	11 – 9	8	7 – 0
OPCODE	b (BIT #)	a	f (FILE #)

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

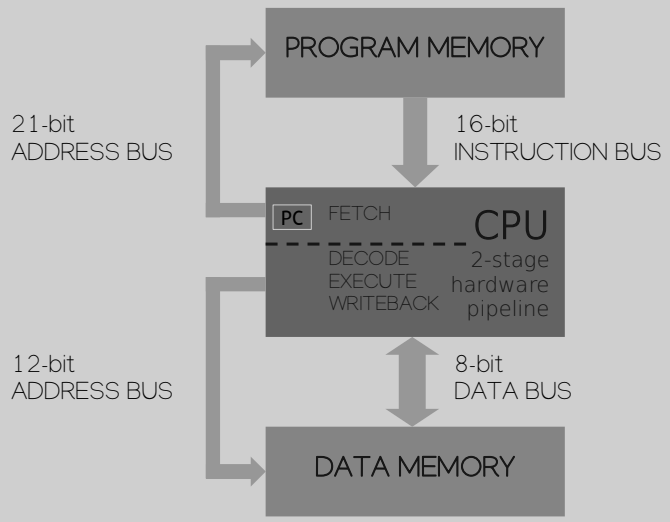
b = 3-bit position of bit in file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

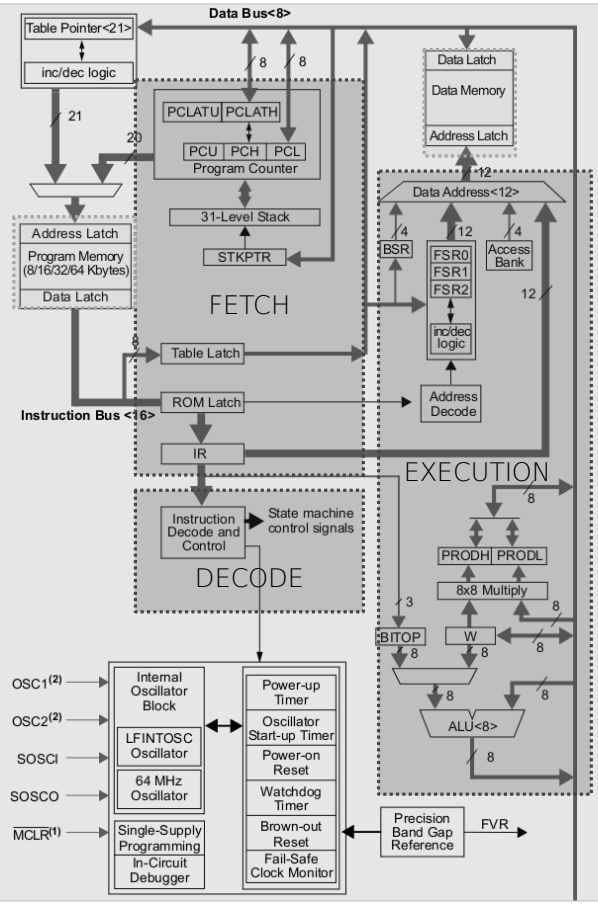
SYSTÈMES EMBARQUES
Assembleur PIC18 - Sommaire

- 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





Programme à traduire

Page 1

```
/* 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;
```

Page 2

```
          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;
}
```

Page 1

```
; 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
```

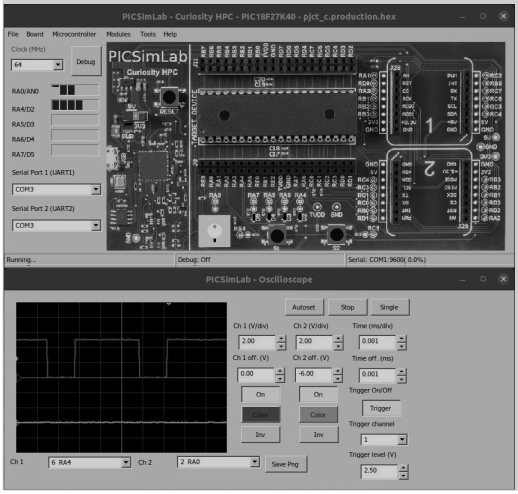
Page 2

```
            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
```

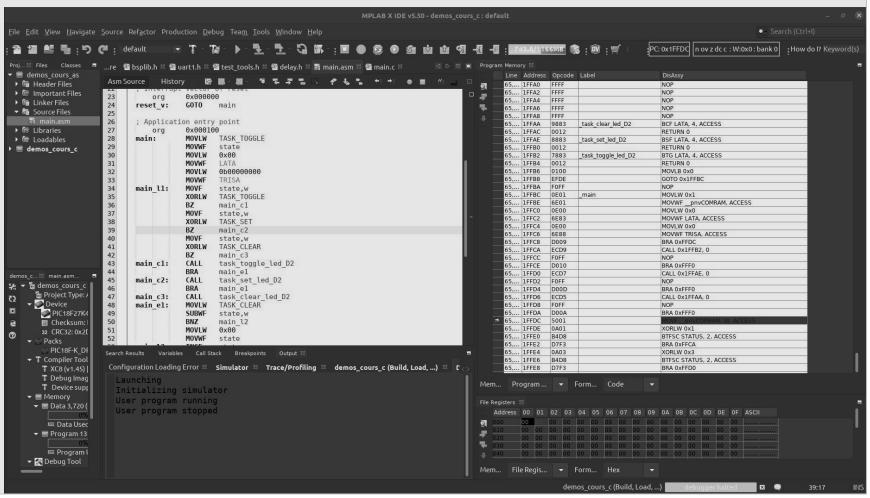

SYSTÈMES EMBARQUES
Solution complète en ASM - Simulateur

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

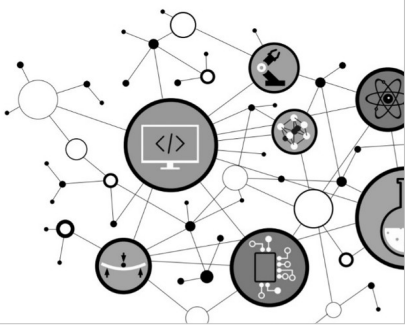


MPLABX Simulator with Debugger

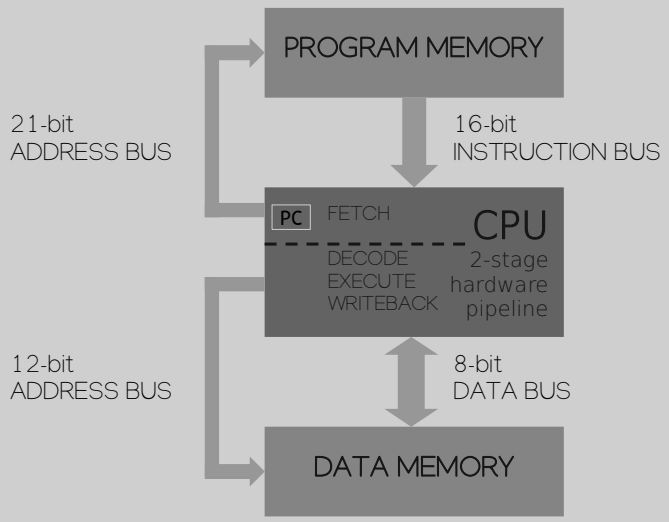


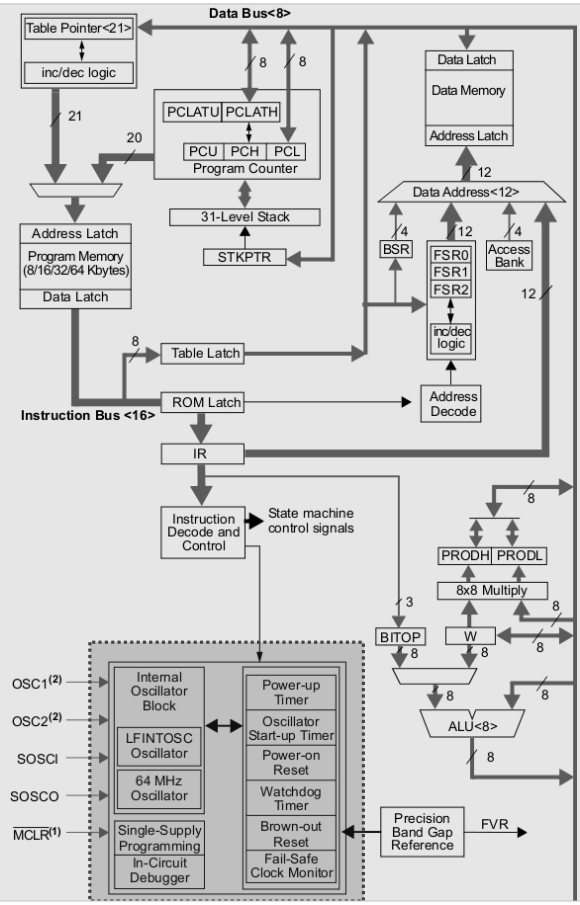
SYSTÈMES EMBARQUES
Assembleur PIC18 - Sommaire

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

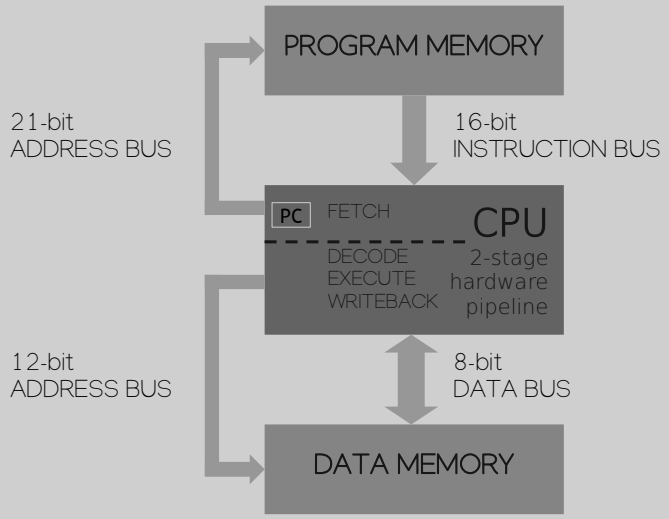


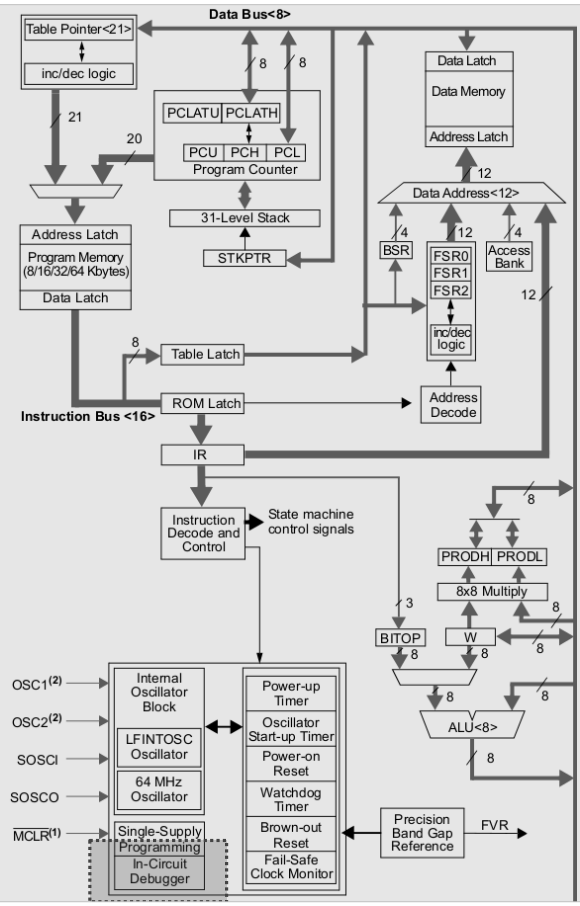
Architecture processeur MCU PIC18





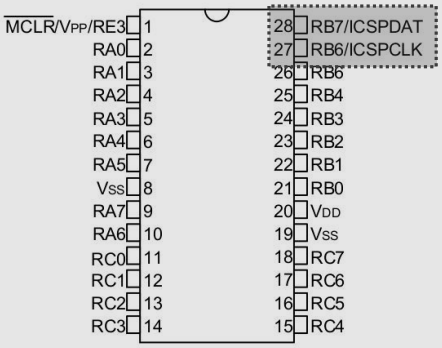
Architecture processeur MCU PIC18



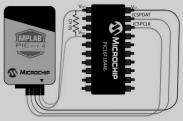


Sans bootloader déjà programmé dans le processeur, nous devons utiliser une sonde JTAG (Join Test Action Group) afin de charger voire debugger le programme depuis l'IDE sur ordinateur vers le MCU cible. Un StarterKit embarque déjà une sonde de programmation à côté du processeur cible de test. Sinon,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

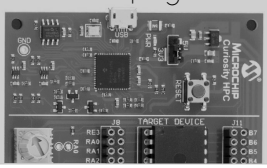
PIC18F27K40
SPDIP 28 pins package



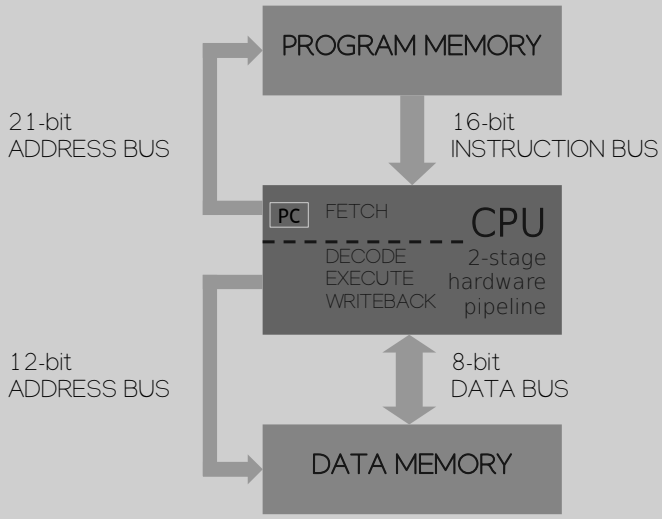
External PICKIT4
JTAG in-circuit programmer/Debugger

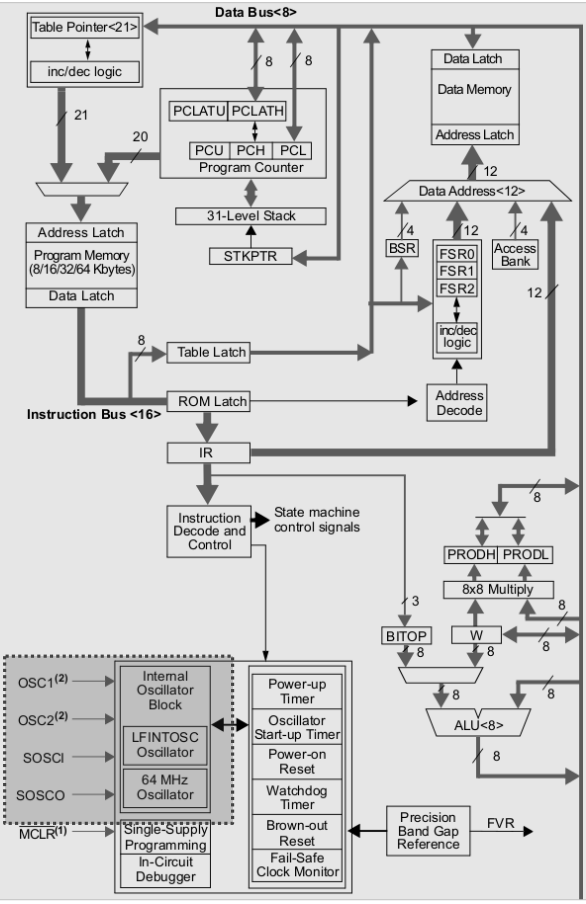


CURIOSITY HPC Starter Kit
with JTAG in-circuit programmer/Debugger



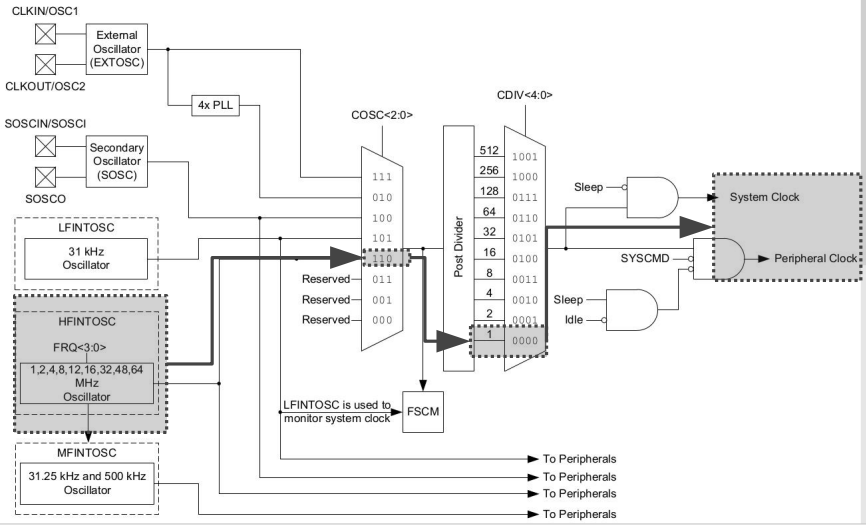
Architecture processeur MCU PIC18





Divers – Oscillateur et horloge de référence

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.



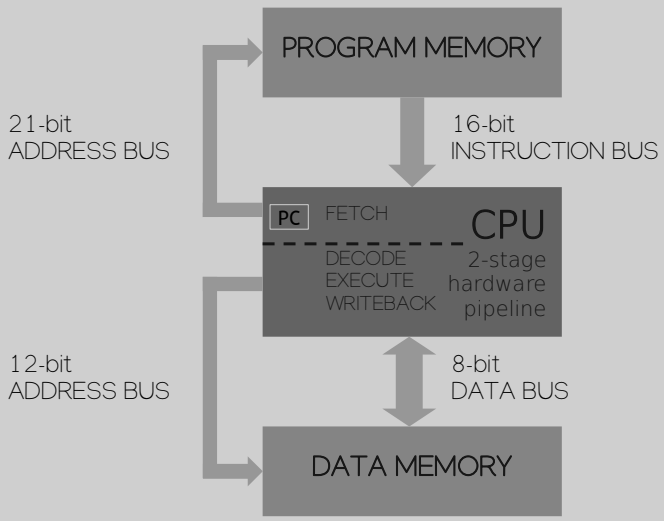
Langage C

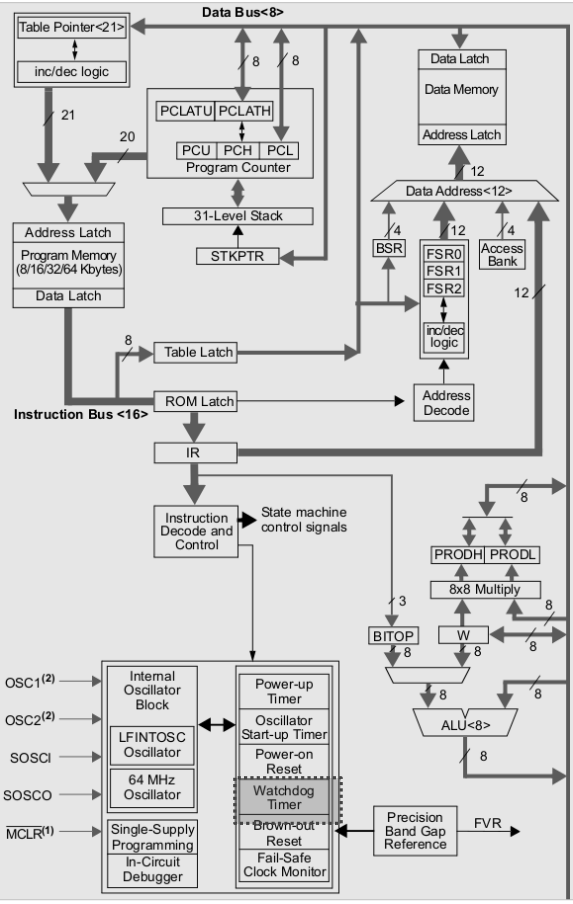
```
#pragma config FEXTOSC = OFF
#pragma config RSTOSC = HFINTOSC_64MHZ
#pragma config MCLRE = EXTMCLR
#pragma config DEBUG = OFF
```

Langage ASM

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

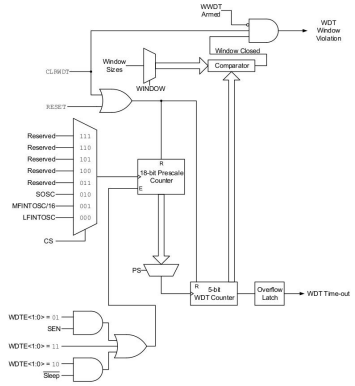
Architecture processeur MCU PIC18





SYSTÈMES EMBARQUES
Divers – Watchdog

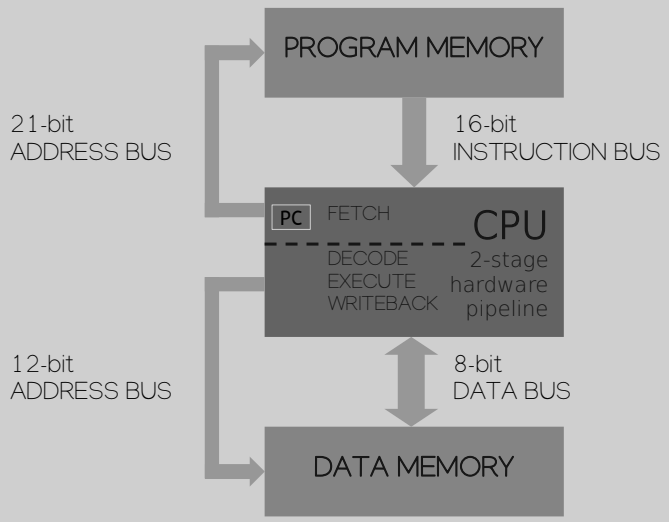
Un watchdog est à ajouter en fin de développement, de test et de validation fonctionnelle d'une application afin d'ajouter une ultime possibilité de redémarrer le programme en cas de défaut grave (application bloquée dans une fonction, boucle infinie, etc). Un Watchdog est un timer pouvant réaliser un 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.

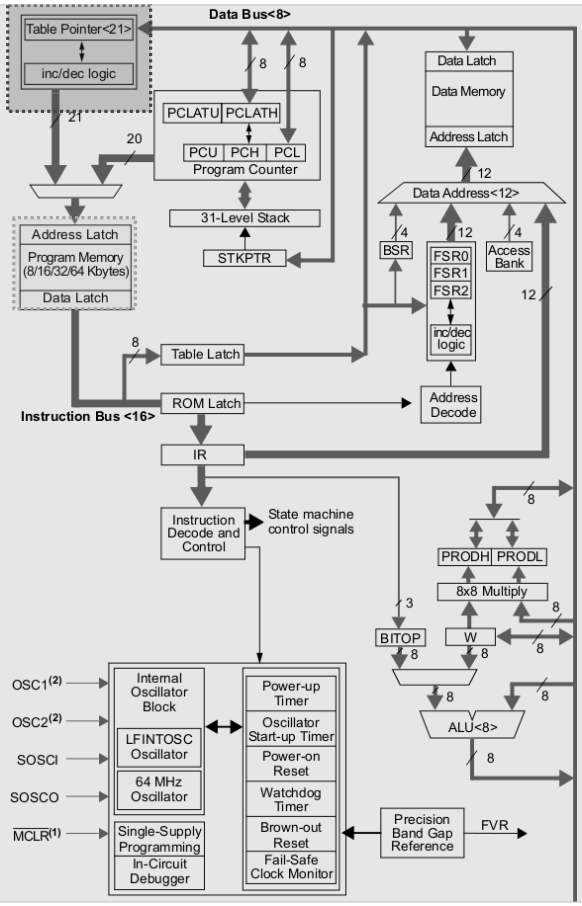


SYSTÈMES EMBARQUES
Divers – Mode veille

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





Divers – Placer des données en mémoire programme

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	

SYSTÈMES EMBARQUES
Divers – Placer des données en mémoire programme

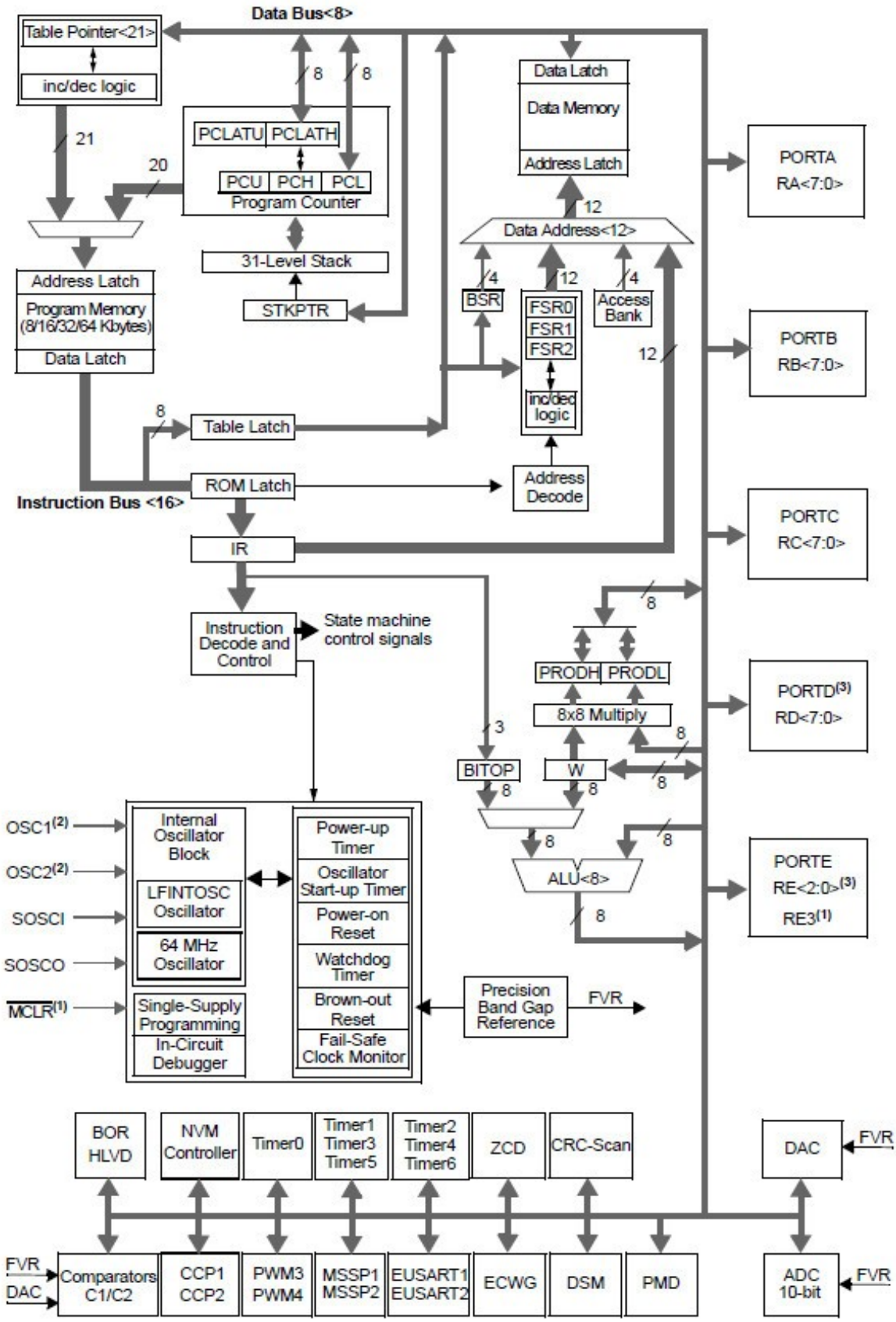
Microchip propose l'accès gratuit à ses outils de développement, notamment ses chaînes de compilation en version LITE. Ces versions ne permettent pas de lever toutes les options d'optimisation à la compilation. Sous XC8 et C18, les versions payantes permettent notamment d'offrir au compilateur C l'accès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

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

SYSTÈMES EMBARQUES
Merci pour votre attention …

ANNEXES

EXTRAITS DE DATASHEETS & GLOSSAIRE


Data Bus<8>
Table Pointer<21>
inc/dec logic
21
20
8
8
PCLATU
PCLATH
PCU
PCH
PCL
Program Counter
31-Level Stack
STKPTR
Address Latch
Program Memory
(8/16/32/64 Kbytes)
Data Latch
Table Latch
ROM Latch
Instruction Bus <16>
IR
Instruction Decode and Control
State machine control signals
Data Latch
Data Memory
Address Latch
12
Data Address<12>
4
12
4
BSR
FSR0
FSR1
FSR2
Access Bank
inc/dec logic
Address Decode
PORTA
RA<7:0>
PORTB
RB<7:0>
PORTC
RC<7:0>
PORTD(3)
RD<7:0>
PORTE
RE<2:0>(3)
RE3(1)
PRODH
PRODL
8x8 Multiply
BITOP
W
ALU<8>
OSC1(2)
OSC2(2)
SOSCI
SOSCO
MCLR(1)
Internal Oscillator Block
LFINTOSC Oscillator
64 MHz Oscillator
Single-Supply Programming
In-Circuit Debugger
Power-up Timer
Oscillator Start-up Timer
Power-on Reset
Watchdog Timer
Brown-out Reset
Fail-Safe Clock Monitor
Precision Band Gap Reference
FVR
BOR
HLVD
NVM Controller
Timer0
Timer1
Timer3
Timer5
Timer2
Timer4
Timer6
ZCD
CRC-Scan
DAC
FVR
FVR
DAC
Comparators C1/C2
CCP1
CCP2
PWM3
PWM4
MSSP1
MSSP2
EUSART1
EUSART2
ECWG
DSM
PMD
ADC 10-bit
FVR

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
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	11da	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	2	1100	ffff	ffff	ffff	None	

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
		f_d (destination) 2nd word		1111	ffff	ffff	ffff		
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	00da	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	
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

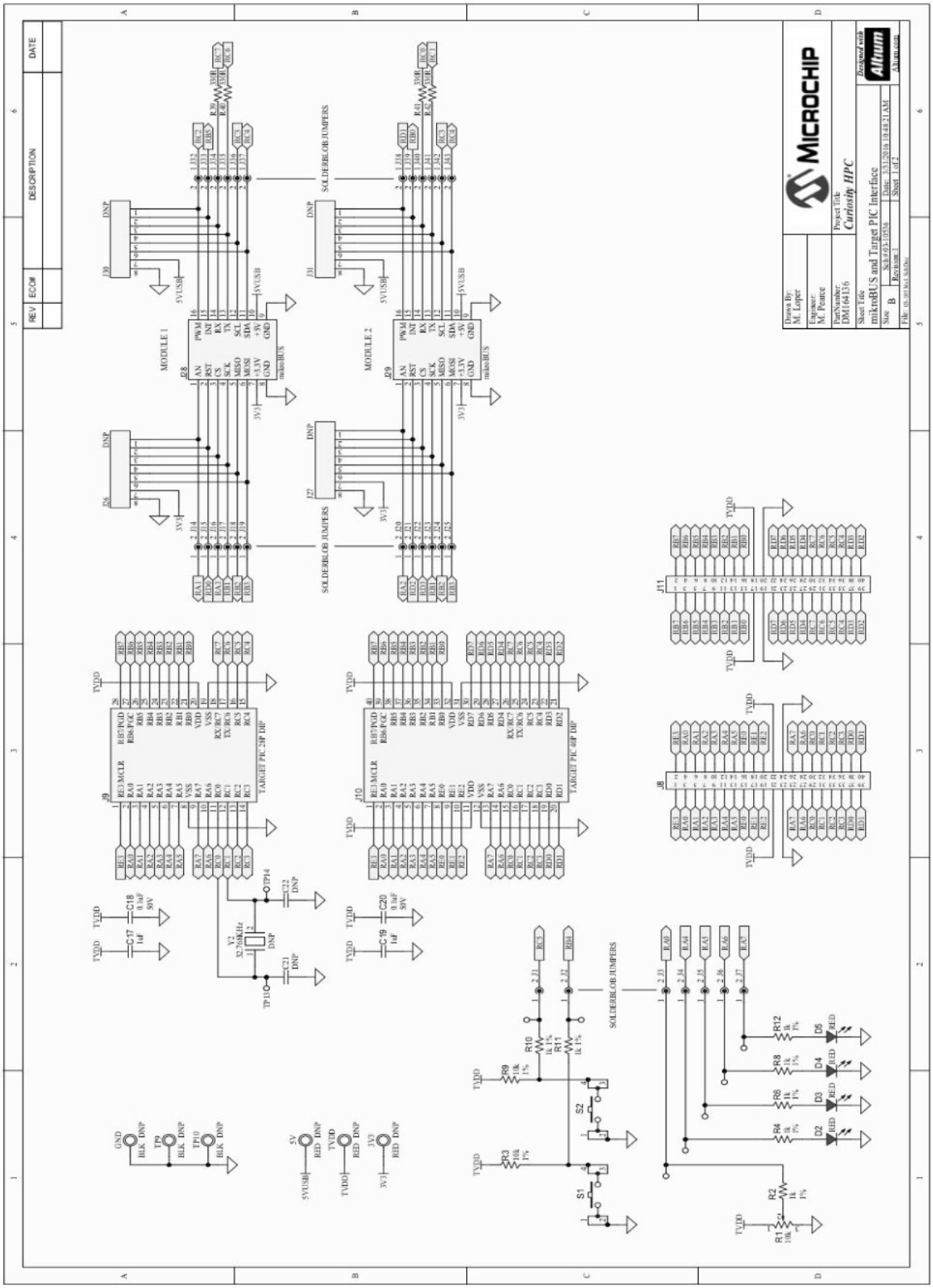
Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, b, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	4
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	k, 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	k	Go to address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		

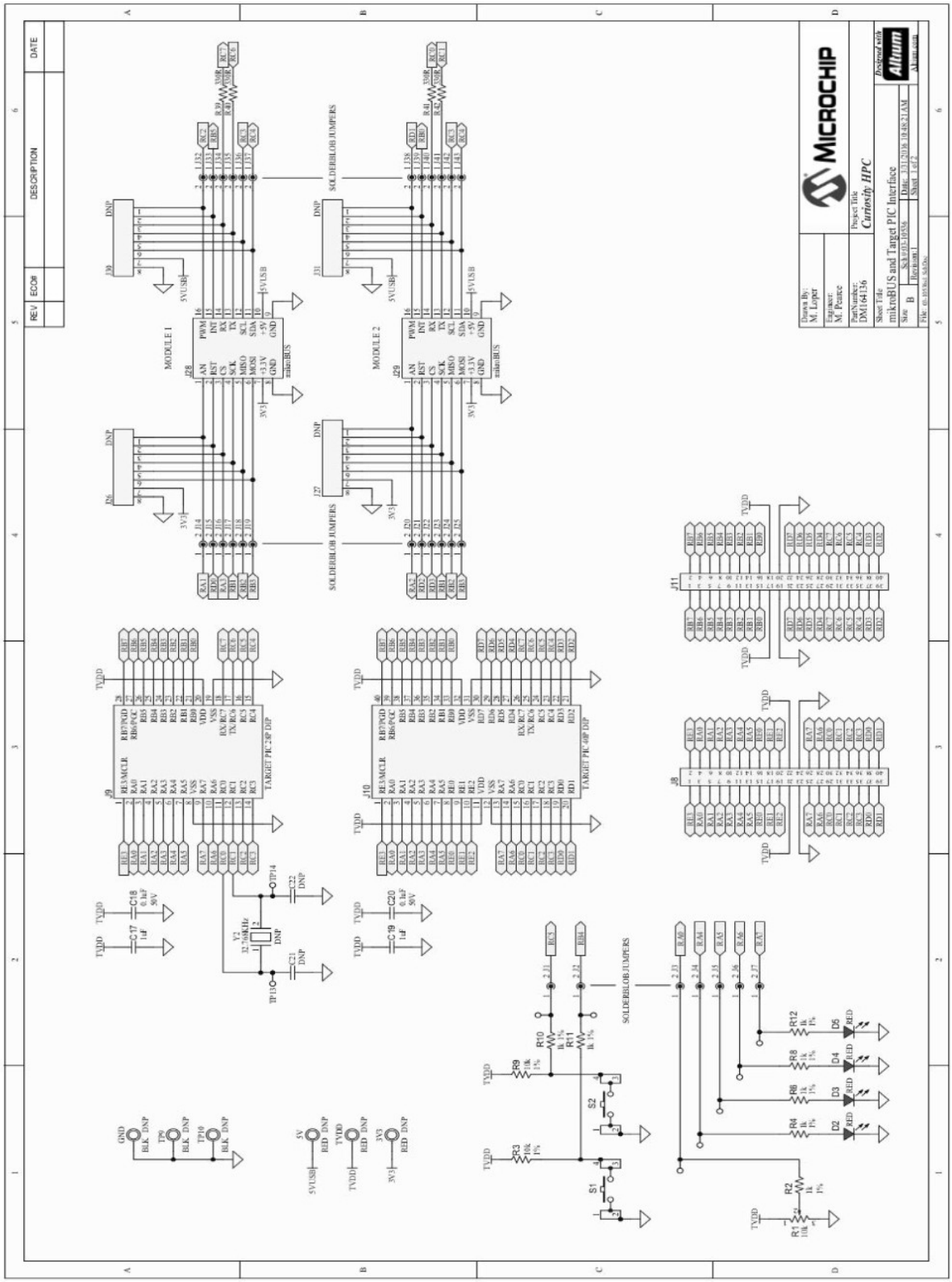
Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	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	$\overline{TO}$, $\overline{PD}$	
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	2	1110	1110	00ff	kkkk	None	
		to FSR(f) 1st word		1111	0000	kkkk	kkkk		
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	

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

Note:

1. When a PORT register is modified as a function of itself (e.g., MOVF PORTB, 1, 0), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.
2. If this instruction is executed on the TMR0 register (and where applicable, 'd' = 1), the prescaler will be cleared if assigned.
3. If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.
4. Some instructions are two-word instructions. The second word of these instructions will be executed as a `NOP` unless the first word of the instruction retrieves the information embedded in these 16 bits. This ensures that all program memory locations have a valid instruction.


MICROCHIP
Project Title
Curiosity HPC
Sheet Title
mikroBUS and Target PIC Interface
Size B
Sheet 1 of 2
Designed with Altium
MODULE 1
MODULE 2
SOLDERBLOB JUMPERS
TARGET PIC 28P DIP
TARGET PIC 40P DIP
mikroBUS


MICROCHIP
Project Title
Curiosity HPC
Sheet Title
mikroBUS and Target PIC Interface
Designed with Altium
MODULE 1
MODULE 2
SOLDERBLOB JUMPERS
TARGET PIC 28P DIP
TARGET PIC 40P DIP

GLOSSAIRE Systèmes Embarqués

A

- ABI : Application Binary Interface
- ADC : Analog to Digital Converter
- ALU : Arithmetic and Logical Unit
- AMD : Advanced Micro Devices
- ANSI : American National Standards Institute
- API : Application Programming Interface
- APU : Accelerrated Processor Unit
- ARM : société anglaise proposant des architectures CPU RISC 32bits
- ASCII : American Standar Code for Information Interchange

B

- BP : Base Pointer
- BSL : Board Support Library
- BSP : Board Support Package

C

- CCS : Code Composer Studio
- CEM : Compatibilité ElectroMagnétique
- CISC : Complex Instruction Set Computer
- CPU : Central Processing Unit
- CSL : Chip Support Library

D

- DAC : Digital to Analog Converter
- DDR : Double Data Rate
- DDR SDRAM: Double Data Rate Synchronous Dynamic Random Access Memory
- DMA : Direct Memory Access
- DSP : Digital Signal Processor
- DSP : Digital Signal Processing

E

- EDMA : Enhanced Direct Memory Access
- EUSART : Enhanced Universal Synchronous Asynchronous Receiver Transmitter
- EMIF : External Memory Interface
- EPIC : Explicitly Parallel Instruction Computing

F

- FPU : Floating Point Unit
- FLOPS : Floating-Point Operations Per Second
- FMA: Fused Multiply-Add

G

- GCC : Gnu Collection Compiler
- GLCD : Graphical Liquid Crytal Display
- GNU : GNU's Not UNIX
- GPIO : General Purpose Input Output
- GPP : General Purpose Processor
- GPU : Graphical Processing Unit

I

- IA-64 : Intel Architecture 64bits
- I2C : Inter Integrated Circuit
- ICC : Intel C++ Compiler
- ICC : Interface Chaise Clavier – les problèmes viennent le plus souvent de cette interface !
- IDE : Integrated Development Environment
- IDMA : Internal Direct memory Access
- IRQ : Interrupt ReQuest
- ISR : Interrupt Software Routine
- ISR : Interrupt Service Routine

L

- L1D : Level 1 Data Memory
- L1I : Level 1 Instruction Memory (idem L1P)
- L1P : Level 1 Program Memory (idem L1I)
- Lx : Level x Memory
- LCD : Liquid Crytal Display
- LRU : Least Recently Used

M

- MAC: Multiply Accumulate
- MCU : Micro Controller Unit
- MIMD : Multiple Instructions on Multiple Data
- MIPS : Mega Instructions Per Second

- MMU : Memory Managment Unit
- MPLABX : MicrochiP LABoratory 10, IDE Microchip
- MPU : Micro Processor Unit ou GPP
- MPU : Memory Protect Unit

O

- OS : Operating System

P

- PC : Program Counter
- PC : Personal Computer
- PIC18 : Famille MCU 8bits Microchip
- PLD : Programmable Logic Device
- POSIX : Portable Operating System Interface, héritage d'UNIX (norme IEEE 1003)
- PPC : Power PC

R

- RAM : Random Access Memory
- RISC : Reduced Instruction Set Computer
- RS232 : Norme standardisant un protocole de communication série asynchrone
- RTOS : Real Time Operating System

S

- SDK : Software Development Kit
- SIMD : Single Instruction Multiple Date
- SIP : System In Package
- SOB : System On Board
- SOC : System On Chip
- SOP : Sums of products
- SP : Stack Pointer
- SP : Serial Port
- SPI : Serial Peripheral Interface
- SRAM : Static Random Access Memory
- SSE : Streaming SIMD Extensions
- STM32 : STMicroelectronics 32bits MCU

T

- TI : Texas Instruments
- TNS : Traitement Numérique du Signal
- TSC : Time Stamp Counter
- TTM : Time To Market

U

- UART : Universal Asynchronous Receiver Transmitter
- USB : Universal Serial Bus

V

- VHDL : VHSIC Hardware Description langage
- VHSIC : Very High Speed Integrated Circuit
- VLIW : Very Long Intruction Word

PRISE DE NOTES

PRISE DE NOTES

PRISE DE NOTES

PRISE DE NOTES

PRISE DE NOTES

PRISE DE NOTES

PRISE DE NOTES