Introduction aux microcontrôleurs et au TI MSP430

Rappel: définition du microcontrôleur

Un **microcontrôleur** est un **circuit intégré** qui rassemble les éléments essentiels d'un ordinateur:

- processeur,
- mémoire,
- unités périphériques,
- interfaces d‘entrée – sortie

Mais les microcontrôleurs se caractérisent par:

- un plus haut degré d'intégration,
- une plus faible consommation électrique,
- une vitesse de fonctionnement plus faible,
- un coût réduit,

par rapport aux microprocesseurs polyvalents utilisés dans les ordinateurs.

Environnements de programmation

Le programme d'un microcontrôleur est généralement appelé *firmware* ou micrologiciel

À l'origine, les microcontrôleurs se programmaient en assembleur uniquement.
Désormais, on utilise de plus en plus des langages de haut niveau, notamment le langage C.
Les compilateurs C pour microcontrôleurs présentent généralement certaines restrictions, et fonctionnalités particulières, liées aux spécificités des microcontrôleurs.

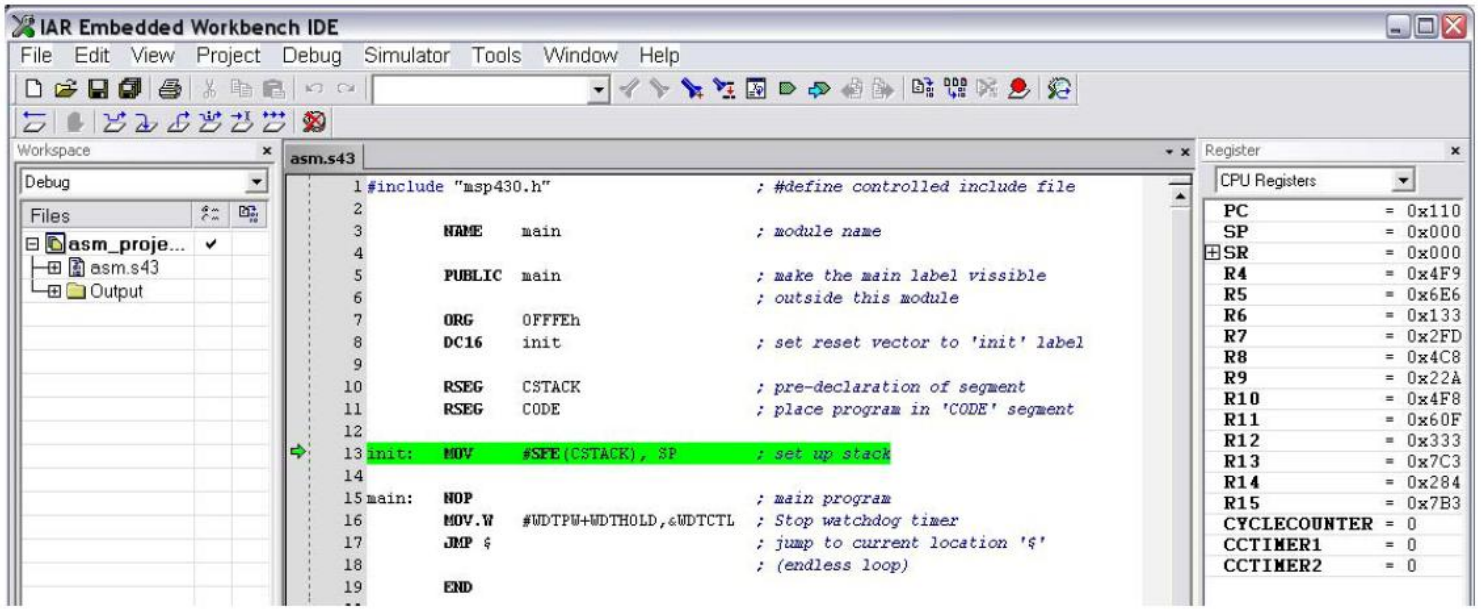
Le programme du microcontroleur est en général développé et compilé sur un PC dans un programme ad hoc (environnement de développement intégré - *IDE*).

Une fois le programme compilé, le fichier binaire doit être envoyé au microcontrôleur.

On utilise en général un programmateur pour microcontrôleurs et souvent également d'EEPROM. Ce type de programmation pourra se faire via le bus de communication standard JTAG ou un autre bus, souvent propriétaire.

Environnements de programmation

Les IDE souvent comprennent un simulateur pour le microcontrôleur.
Les développeurs peuvent ainsi analyser le comportement du microcontrôleur et du programme sur un PC, comme s'ils s'agissait du composant réel.



Un simulateur montre l'état interne du processeur, ainsi que celui de ses sorties.
Bien que la plupart des simulateurs ne proposent pas de simuler les autres composants d'un système, ils permettent de spécifier les entrées à volonté.
Cela facilite donc l'analyse et la résolution des problèmes.

Familles de microcontrôleurs

la famille Atmel AT91 ;
la famille Atmel AVR ;
le C167 de Siemens / Infineon ;
la famille Hitachi H8 ;
la famille Intel 8051, certains processeurs récents utilisent un cœur 8051, qui est complété par divers périphériques
l'Intel 8085, à l'origine conçu pour être un microprocesseur, a en pratique souvent été utilisé en tant que microcontrôleur
le Freescale 68HC11 ;
la famille Freescale 68HC08 ;
la famille Freescale 68HC12 ;
la famille des PIC de Microchip ;
la famille des dsPIC de Microchip ;
la famille des ST6, ST7, ST10, STR7, STR9 de STMicroelectronics ;
la famille ADuC d'Analog Devices ;
la famille PICBASIC de *Comfile Technology* ;
la famille MSP430 de Texas Instruments ;
la famille 8080, dont les héritiers sont le microprocesseur Zilog Z80 (désormais utilisé en tant que contrôleur dans l'embarqué)
la famille *PSoC* de *Cypress* ;
la famille *LPC21xx ARM7-TDMI* de Philips ;
la famille V800 de NEC ;
la famille K0 de NEC.

Texas Instruments MSP430

Ultra-low Power + High-Performance

- 0.1µA power down
- 0.8µA standby mode
- 250µA / 1MIPS
- <1µs clock start-up
- Zero-power BOR
- <50nA port leakage

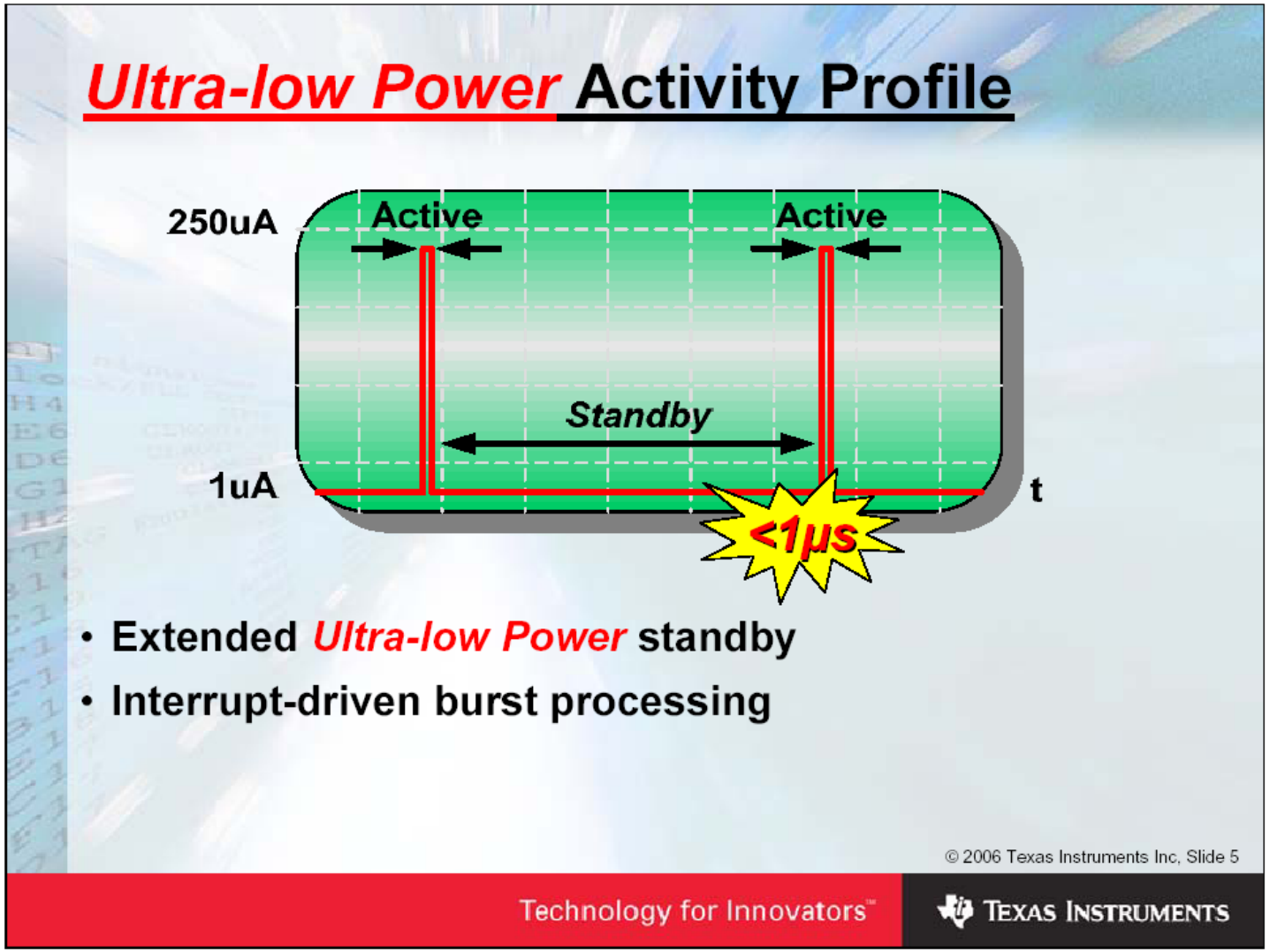
- Modern 16-bit RISC CPU
- 1K to 128KB+ ISP Flash
- 14- to 100-pin options
- Intelligent peripherals boost performance
- Embedded emulation

Technology for Innovators™ Texas Instruments

- Construit autour d'un CPU 16 bits, le **MSP430** a été conçu pour des applications embarquées à basse consommation et à faible coût.
- Il est particulièrement adapté aux applications sans-fil fonctionnant sur batteries.
- Son mode de fonctionnement en attente consomme moins de 1 microampère.

- Vidéo: http://www.youtube.com/watch?v=ZxGZIiyyxrM

- Dans les microcontrôleurs 16 bits, le MSP430 vise les applications *très basse consommation*



Selected Package Options for MSP430 Devices

20-pin PW (TSSOP)	20-pin DGV (TVSOP)	28-pin PW (TSSOP)	48-pin DL (SSOP)	64-pin PM (QFP)
(6,60) (6,60)	(5,10) (6,60)	(9,80) (6,60)	(16,00) (10,67)	(12,20) (12,20)

20-pin DW (SOP)	-pin DW (SOP)	56-pin DL (SSOP)	100-pin PZ (LQFP)
(12,95) (10,65)	(10,65)	(18,54) (10,67)	(16,20) (16,20)

- Le composant est décliné en une série de configurations comprenant les périphériques usuels:

  - Le composant est décliné en une série de configurations comprenant les périphériques usuels :
  - convertisseurs A/D 10/12/14/16 bits,
  - convertisseurs D/A 12 bits,
  - comparateurs,
  - interfaces USART, SPI, I2C
  - pilote LCD
  - Watchdog
  - multiplicateur hardware
  - DMA
  - oscillateur interne,
  - etc.

- Le MSP430 fonctionne à des fréquences de l'ordre de 8 MHz, 16 MHz pour les modèles les plus récents.
- Il a toutefois des limitations qui l'empêchent d'être utilisé dans des systèmes embarqués plus complexes.
- Par exemple, il n'a pas de bus mémoire externe (qui permettrait d'accéder efficacement à des mémoires RAM / ROM externes) et sa capacité de mémoire (8K de RAM, 120 K de flash pour les mieux dotés) peut se révéler insuffisante pour des applications qui demandent de grandes tables de données.

Modèle de programmation

- On appelle modèle de programmation, l'ensemble des éléments nécessaires à l'utilisateur pour la programmation d'un processeur :

  - modes d'adressage,

  - jeu d'instructions,

  - registres internes (ceux accessibles au programmeur).

- Le nombre de modes d'adressage possibles, le nombre d'instructions et le nombre de registres caractérisent le type de processeur : CISC ou RISC.

  CISC : *Complex Instruction Set Computer*

  RISC : *Reduced Instruction Set Computer*

CISC/RISC : un débat des années 90 …

Performance d'un processeur = temps pour effectuer une tâche

ou:

Temps / tâche = nombre instruction / tâche x no cycles / instr x Tps / cycle

Approche RISC : on cherche à réduire le **nombre de cycles par instruction** (idéalement un cycle par instruction).

Cela ne peut être tenu que par un nombre restreint d'instructions et de modes d'adressage simples.

-> nombre instructions par tâche augmente, mais

-> décodage plus rapide -> le **temps par cycle** diminue

(et surtout : conception plus simple, le prix diminue…)

- **Approche CISC**: on cherche à réduire le nombre de **instructions par tâche**.
- Cela entraîne un jeu d'instruction toujours plus étendu, des instructions de plus en plus complexes, des modes d'adressage plus complexes également.
- Dans cette voie, la palme revient probablement aux processeurs de Motorola de la série 680X0 (du 68020 au 68060) :

move.l ([$200,a6], d2*4, $50), d3

a6 (registre d'adresse) gère une zone de variables

a6+200 : permet d'accéder à l'adresse d'un pointeur : zone de 4 octets qui contient une autre adresse (c'est un mode d'adressage indirect : [ ])

Cette adresse est celle d'une structure particulière

+$50 : accède un élément de cette structure.

D2, registre de données, contient l'indice d'un élément d'un tableau. Il faut donc se déplacer de d2*(taille de chaque élément) pour atteindre cet élément.

Le modèle de programmation du MSP430

* instructions

un cœur RISC :

27 instructions de base

+ 24 instructions « émulées »

* Modes d'adressage

7 modes d'adressage pour l'opérande source

4 modes d'adressage pour l'opérande destination

Mode d'adressage	Opérande source	Opérande destination
Registre (ou direct par registre)	✓	✓
Indexé	✓	✓
Symbolique	✓	✓
Absolu	✓	✓
Indirect	✓	
Indirect auto-incrémenté	✓	
immédiat	✓	

Tab.: Modes d'adressage

As/Ad	Mode d'adressage	Syntaxe	Description
00/0	Mode registre	Rn	Le contenu du registre est l'opérande
01/1	Mode indexé	X(Rn)	(Rn + X) pointe vers l'opérande X est stocké dans le mot suivant
01/1	Mode symbolique	ADDR	L'opérande est le contenu à l'adresse PC+ADDR
01/1	Absolute mode	&ADDR	L'opérande est le contenu à l'adresse absolue ADDR
10/-	Mode registre indir.	@Rn	Rn est utilisé comme pointeur vers l'opérande
11/-	Indirect auto inc.	@Rn+	Rn est utilisé comme pointeur vers l'opérande et est incrémenté après l'opération
11/-	Mode immédiat	#N	L'opérande est la constant immédiate N.

PC = Program Counter

Utilisation typique des modes d'adressage

Register Mode	`mov.w R10,R11` **Single cycle**
Indexed Mode	`mov.w 2(R5),6(R6)` **Table processing**
Symbolic Mode	`mov.w EDE,TONI` **Easy to read code, PC relative**
Absolute Mode	`mov.w &EDE,&TONI` **Directly access any memory location**
Indirect Register Mode	`mov.w @R10,0(R11)` **Access memory with pointers**
Indirect Autoincrement	`mov.w @R10+,0(R11)` **Table processing**
Immediate Mode	`mov.w #45h,&TONI` **Unrestricted constant values**

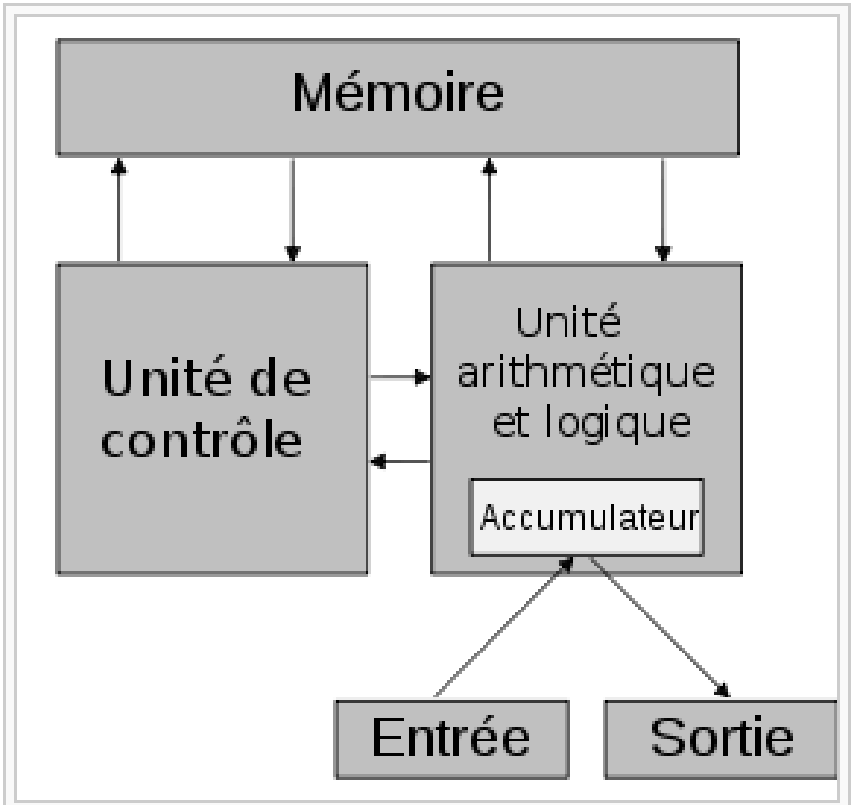
- La CPU du MSP430 a une **architecture de von Neumann**, avec un espace unique pour adresser instructions et données.
- La mémoire est adressée par bytes et pairs de bytes sont combinées pour faire des words de 16 bits.

- Le processeur contient 16 registres de 16 bits.
  - R0 est le *program counter*,
  - R1 est le *stack pointer*,
  - R2 est le *status register*
  - R3 est un registre spécial appelé le *constant generator*, et donnant accès à 6 constantes utilisées fréquemment sans devoir utiliser d'autres opérands.
  - R4 à R15 disponibles pour usage général.

Architecture d'un processeur von Neumann

L'architecture, dite architecture de von Neumann, est un modèle pour un ordinateur qui utilise une structure de stockage unique pour conserver à la fois les instructions et les données requises ou générées par le calcul.

De telles machines sont aussi connues sous le nom d'*ordinateurs à programme stocké en mémoire*. La séparation entre le stockage et le processeur est implicite dans ce modèle.

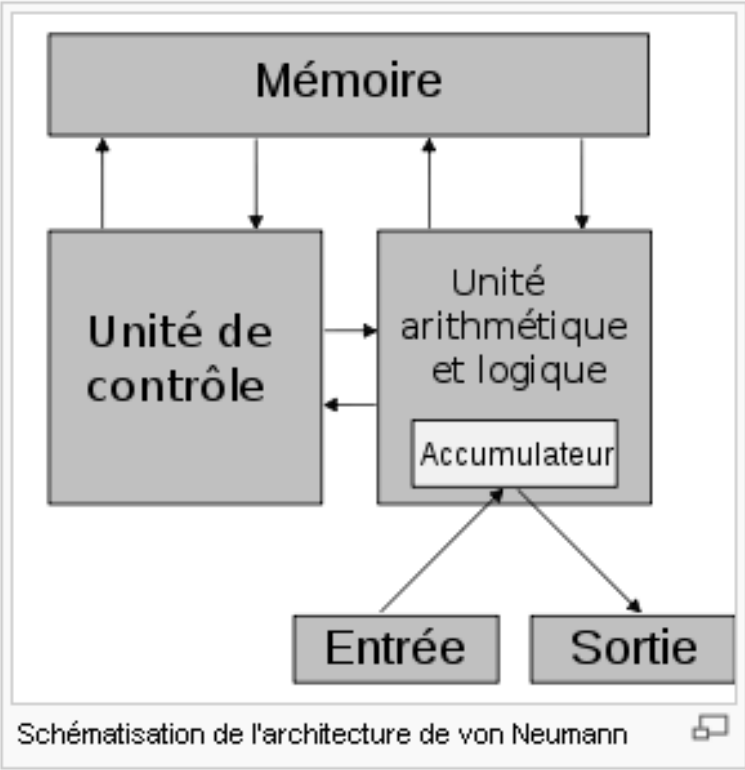


Schématisation de l'architecture de von Neumann

Architecture d'un processeur von Neumann

L'architecture de von Neumann décompose l'ordinateur en 4 parties distinctes

1. L'unité arithmétique et logique (ALU en anglais) ou unité de traitement: son rôle est d'effectuer les opérations de base.
2. L'unité de contrôle, chargée du séquençage des opérations.
3. La mémoire qui contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise entre mémoire volatile (programmes et données en cours de fonctionnement) et mémoire permanente (programmes et données de base de la machine).
4. Les dispositifs d'entrée-sortie, qui permettent de communiquer avec le monde extérieur.



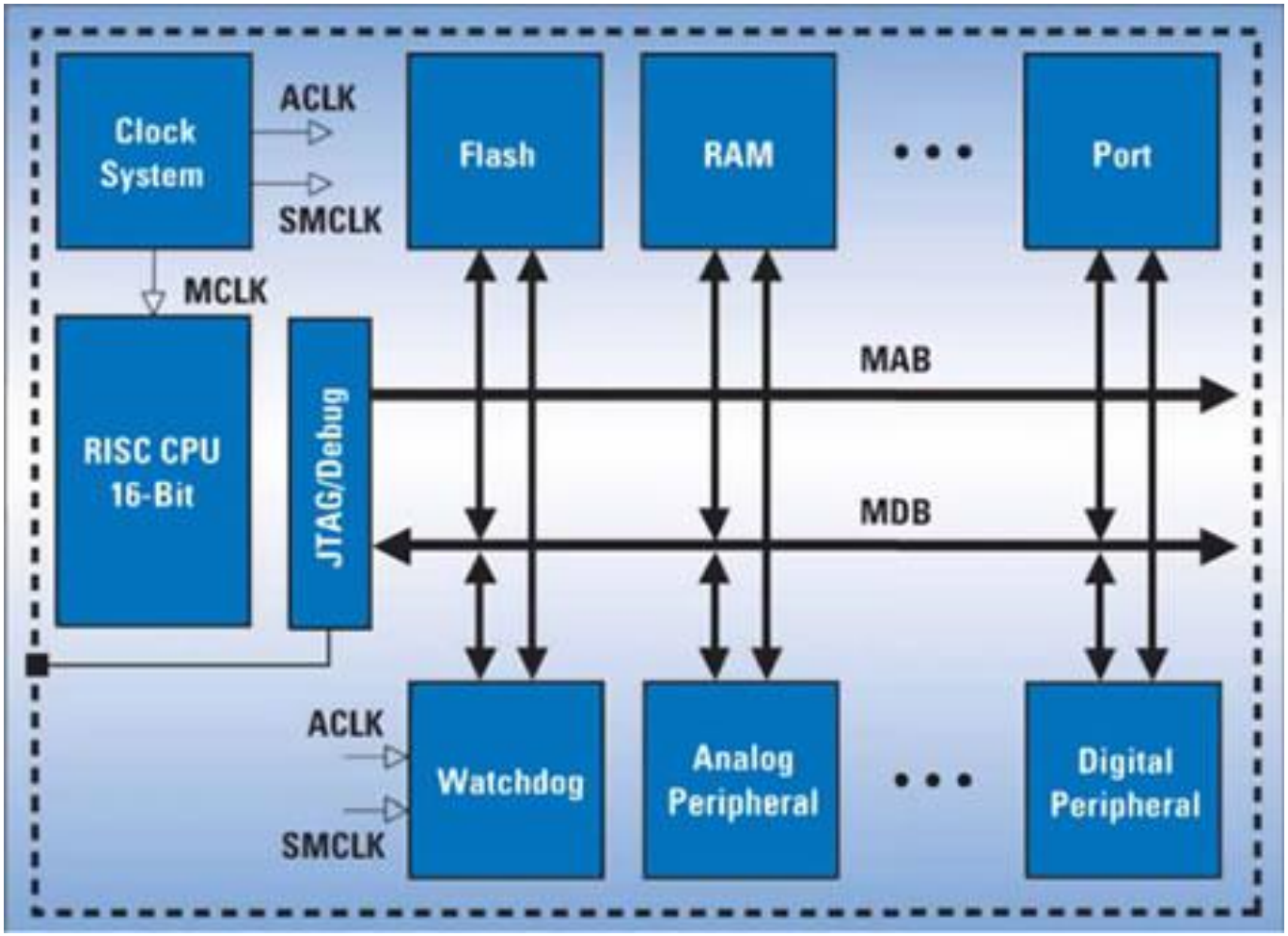
Schématisation de l'architecture de von Neumann

Architecture d'un processeur von Neumann

Modification des instructions

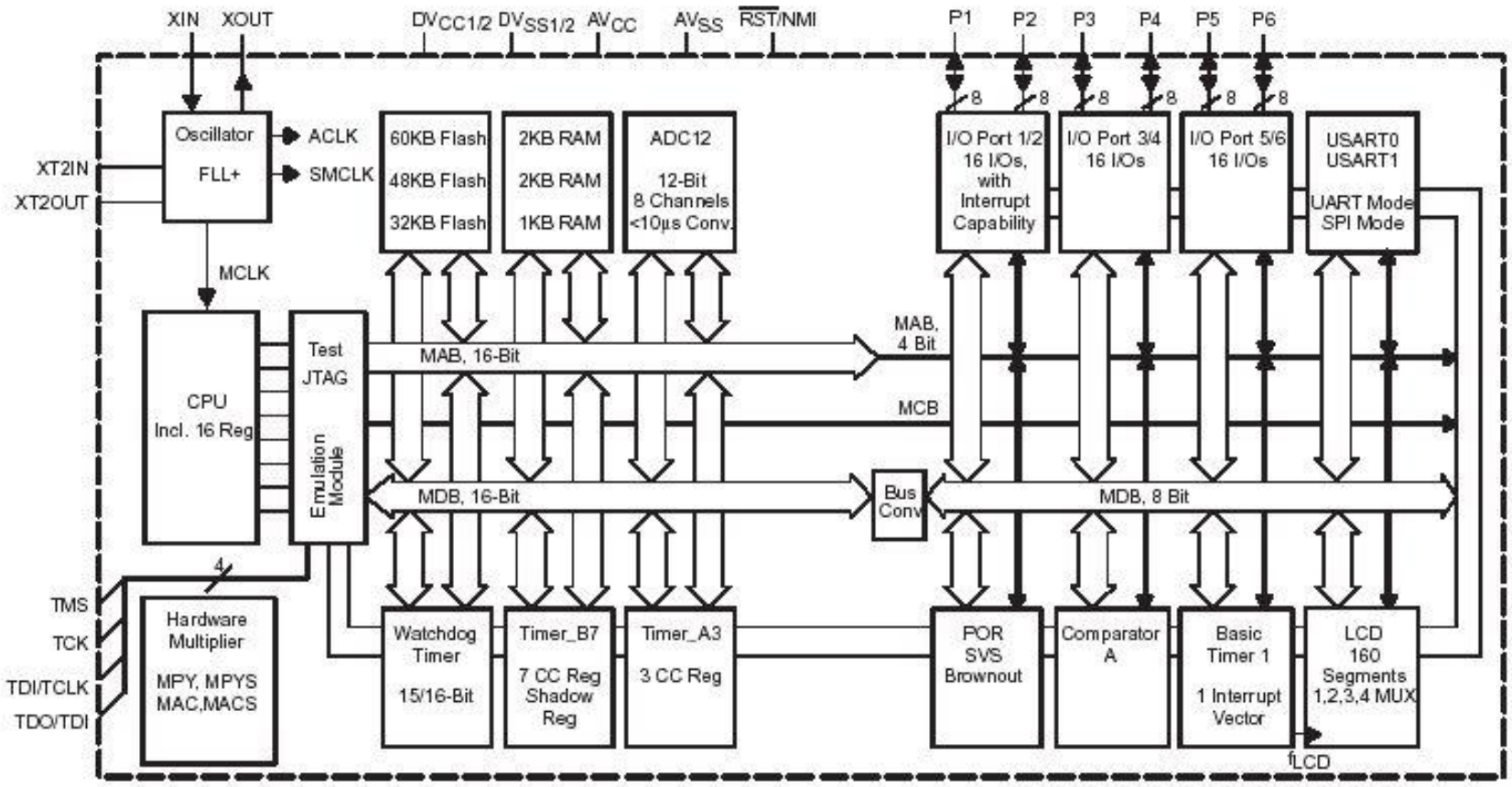
- En traitant les instructions de la même façon que les données, un ordinateur à programme stocké en mémoire peut facilement modifier les instructions. Une motivation importante pour une telle facilité était le besoin pour un programme d'incrémenter ou modifier le champ des adresses des instructions (pour faire des boucles par exemple). Cette motivation est devenue moins importante en même temps que l'utilisation de registres d'index et de l'adressage indirect devenait une caractéristique standard des processeurs.
- L'architecture actuelle des ordinateurs rend inutile la modification à faible échelle des instructions du programme — son « code » — car cela rendrait inefficace les techniques de gestion de la mémoire et du pipeline dans le processeur. Cette pratique est donc à ce jour dépréciée. Bien sûr, à une plus large échelle, la possibilité de traiter des instructions comme étant des données est ce qui permet l'écriture de **compilateurs**.
- C'est aussi une caractéristique qui peut être exploitée par les **virus** lorsqu'ils ajoutent une copie d'eux-mêmes dans le code de programmes existants. Le problème de la réplication de code non autorisée peut être contourné par l'utilisation d'un système de protection de la mémoire et, en particulier, par le gestionnaire de la mémoire virtuelle.

Architecture générale d'un MSP430 :



MAB = memory address bus **MDB = memory data bus**

Architecture (détails) d'un MSP430



- Les instructions font 16 bits, suivis par (au maximum) deux *words* de 16-bit.
- Les modes d'adressage sont spécifiés par le champ *As* (2 bits).
- B/W = byte / word

MSP430 instruction set

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction
0	0	0	1	0	0	opcode			B/W	As		register				Single-operand arithmetic
0	0	0	1	0	0	0	0	0	B/W	As		register				**RRC** Rotate right through carry
0	0	0	1	0	0	0	0	1	0	As		register				**SWPB** Swap bytes
0	0	0	1	0	0	0	1	0	B/W	As		register				**RRA** Rotate right arithmetic
0	0	0	1	0	0	0	1	1	0	As		register				**SXT** Sign extend byte to word
0	0	0	1	0	0	1	0	0	B/W	As		register				**PUSH** Push value onto stack
0	0	0	1	0	0	1	0	1	0	As		register				**CALL** Subroutine call; push PC and move source to PC
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	**RETI** Return from interrupt; pop SR then pop PC
0	0	1	condition			10-bit signed offset										Conditional jump; PC = PC + 2×offset
0	0	1	0	0	0	10-bit signed offset										**JNE/JNZ** Jump if not equal/zero
0	0	1	0	0	1	10-bit signed offset										**JEQ/JZ** Jump if equal/zero
0	0	1	0	1	0	10-bit signed offset										**JNC/JLO** Jump if no carry/lower
0	0	1	0	1	1	10-bit signed offset										**JC/JHS** Jump if carry/higher or same
0	0	1	1	0	0	10-bit signed offset										**JN** Jump if negative
0	0	1	1	0	1	10-bit signed offset										**JGE** Jump if greater or equal
0	0	1	1	1	0	10-bit signed offset										**JL** Jump if less
0	0	1	1	1	1	10-bit signed offset										**JMP** Jump (unconditionally)

opcode				source	Ad	B/W	As	destination	Two-operand arithmetic
0	1	0	0	source	Ad	B/W	As	destination	**MOV** Move source to destination
0	1	0	1	source	Ad	B/W	As	destination	**ADD** Add source to destination
0	1	1	0	source	Ad	B/W	As	destination	**ADDC** Add source and carry to destination
0	1	1	1	source	Ad	B/W	As	destination	**SUBC** Subtract source from destination (with carry)
1	0	0	0	source	Ad	B/W	As	destination	**SUB** Subtract source from destination
1	0	0	1	source	Ad	B/W	As	destination	**CMP** Compare (pretend to subtract) source from destination
1	0	1	0	source	Ad	B/W	As	destination	**DADD** Decimal add source to destination (with carry)
1	0	1	1	source	Ad	B/W	As	destination	**BIT** Test bits of source AND destination
1	1	0	0	source	Ad	B/W	As	destination	**BIC** Bit clear (dest &= ~src)
1	1	0	1	source	Ad	B/W	As	destination	**BIS** Bit set (logical OR)
1	1	1	0	source	Ad	B/W	As	destination	**XOR** Exclusive or source with destination
1	1	1	1	source	Ad	B/W	As	destination	**AND** Logical AND source with destination (dest &= src)

Registres

- Il y a 16 registres à 16-bit .
  - R0 program counter,
  - R1 stack pointer,
  - R2 status register,
  - R3 constant zero register
  - R4 à R15 disponibles.

- Il y a quatre modes d'adressage, spécifié par le champ *As* (2 bits).
- Les mode de *indexed addressing* ajoutent un *word* de 16-bit à l'instruction.

MSP430 addressing modes

As	Register	Syntax	Description
00	*n*	**R*n***	Register direct. The operand is the contents of R*n*.
01	*n*	***x*(R*n*)**	Indexed. The operand is in memory at address R*n*+*x*.
10	*n*	**@R*n***	Register indirect. The operand is in memory at the address held in R*n*.
11	*n*	**@R*n*+**	Indirect autoincrement. As above, then the register is incremented by 1 or 2.
Addressing modes using R0 (PC)			
01	0 (PC)	***LABEL***	Symbolic. x(PC) The operand is in memory at address PC+x.
11	0 (PC)	***#x***	Immediate. @PC+ The operand is the next word in the instruction stream.
Addressing modes using R2 (SP) and R3 (CG), special-case decoding			
01	2 (SR)	***&LABEL***	Absolute. The operand is in memory at address *x*.
10	2 (SR)	**#4**	Constant. The operand is the constant 4.
11	2 (SR)	**#8**	Constant. The operand is the constant 8.
00	3 (CG)	**#0**	Constant. The operand is the constant 0.
01	3 (CG)	**#1**	Constant. The operand is the constant 1. There is no index word.
10	3 (CG)	**#2**	Constant. The operand is the constant 2.
11	3 (CG)	**#-1**	Constant. The operand is the constant -1.

Multiplication 16 bits x 16 bits et addition (accumulation)

Le **multiplicateur hardware** est un périphérique, il ne fait donc pas partie de l'unité de traitement du MSP430.

Description des opérations

- multiplication de nombres non signés
- multiplication de nombres signés
- multiplication et accumulation non signée
- multiplication et accumulation signée
- 16×16 bits, 16×8 bits, 8×16 bits, 8×8 bits

Le multiplicateur utilise 8 registres:

Address	Name	Function
0x130	MPY	Operand1 for unsigned multiply
0x132	MPYS	Operand1 for signed multiply
0x134	MAC	Operand1 for unsigned multiply-accumulate
0x136	MACS	Operand1 for signed multiply-accumulate
0x138	OP2	Second operand for multiply operation
0x13a	ResLo	Low word of multiply result
0x13c	ResHi	High word of multiply result
0x13e	SumExt	Carry out of multiply-accumulate

Multiplication de deux nombres entiers non signés 1234h x ABCDh = C374FA4h

Multiplication entière non signée 4660·43981 = 204951460

```
MOV #01234h,&MPY ; premier opérande
MOV #0ABCDh,&OP2 ; second opérande
```

} RESHI = 0C37h RESLO = 4FA4h

Multiplication de deux nombres entiers signés 1234h x ABCDh = FA034FA4h

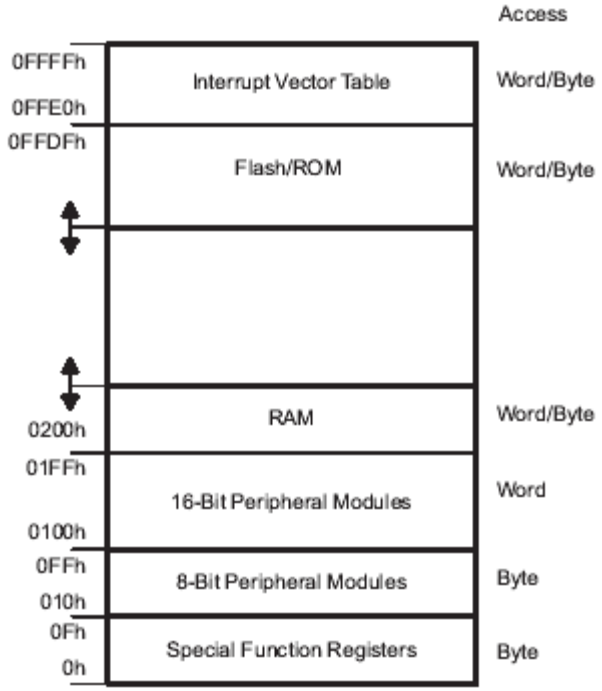
Multiplication entière signée 4660·(-21555)=-100446300

```
MOV #01234h,&MPYS ; premier opérande
MOV #05678h,&OP2 ; second opérande
```

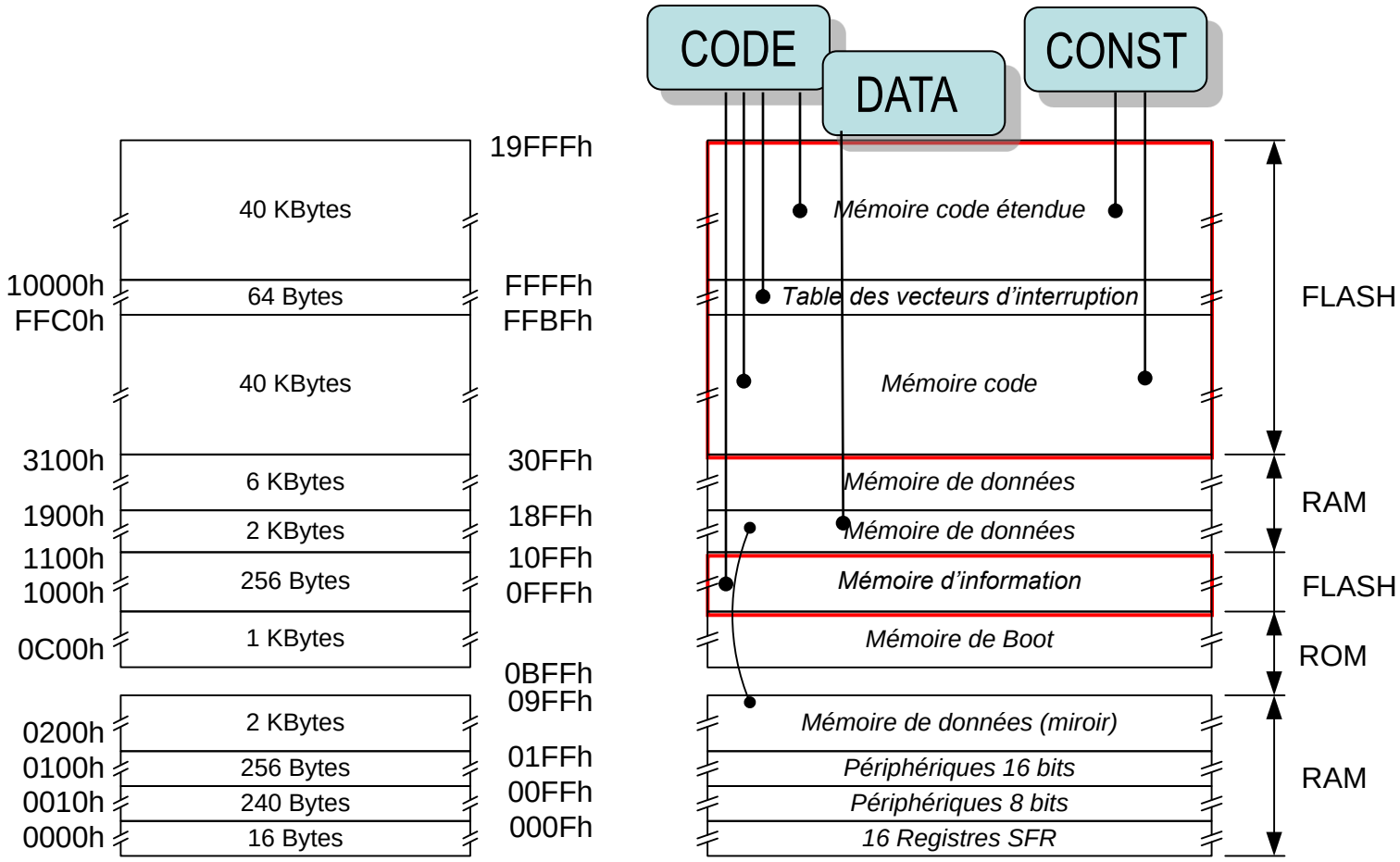
} SUMEXT FFFFh RESHI = FA03h RESLO = 4FA4h

Mémoire

- L'architecture von-Neumann du MSP430 a un seul espace d'adressage partagé entre les registres spéciaux (SFRs), les périphériques, la RAM, et la Flash/ROM.
- Les accès code sont toujours faits sur des adresses paires.
- Les données peuvent être accédées comme *bytes* ou *words*.
- L'espace adressable de mémoire est de 64Kb avec des extensions futures prévues.



Segmentation de la mémoire (cas du MSP430FG4617)



Interruptions

Une interruption est un signal demandant au processeur de suspendre temporairement l'exécution du programme courant afin d'effectuer des opérations particulières.

Ce mécanisme permet d'implémenter une réaction à une sollicitation en respectant les exigences suivantes

⇨ offrir un délai de réponse très bref,

⇨ programmation indépendante du code en cours d'exécution.

Le système d'interruption est le dispositif incorporé au séquenceur qui détecte les signaux d'interruption. Ces signaux arrivent de façon asynchrone, à n'importe quel moment, mais ils ne sont pris en compte qu'à la fin de l'opération en cours.

Origine des interruptions

Les interruptions peuvent être déclenchées :

soit par un composant extérieur au processeur

- ⇨ requête d'interruption (Interrupt ReQuest, IRQ), signalée via une ou plusieurs
- ⇨ broches d'entrée/sortie configurables ou non,

soit par un périphérique interne au composant (cas des microcontrôleurs ou des DSP)

- ⇨ Interruption initiée par un timer, convertisseur A/N, watchdog, …

soit la CPU

- ⇨ exception d'exécution ou dépassement lors d'un calcul arithmétique,
- ⇨ interruption logicielle
- ⇨ …

Mécanisme d'interruption

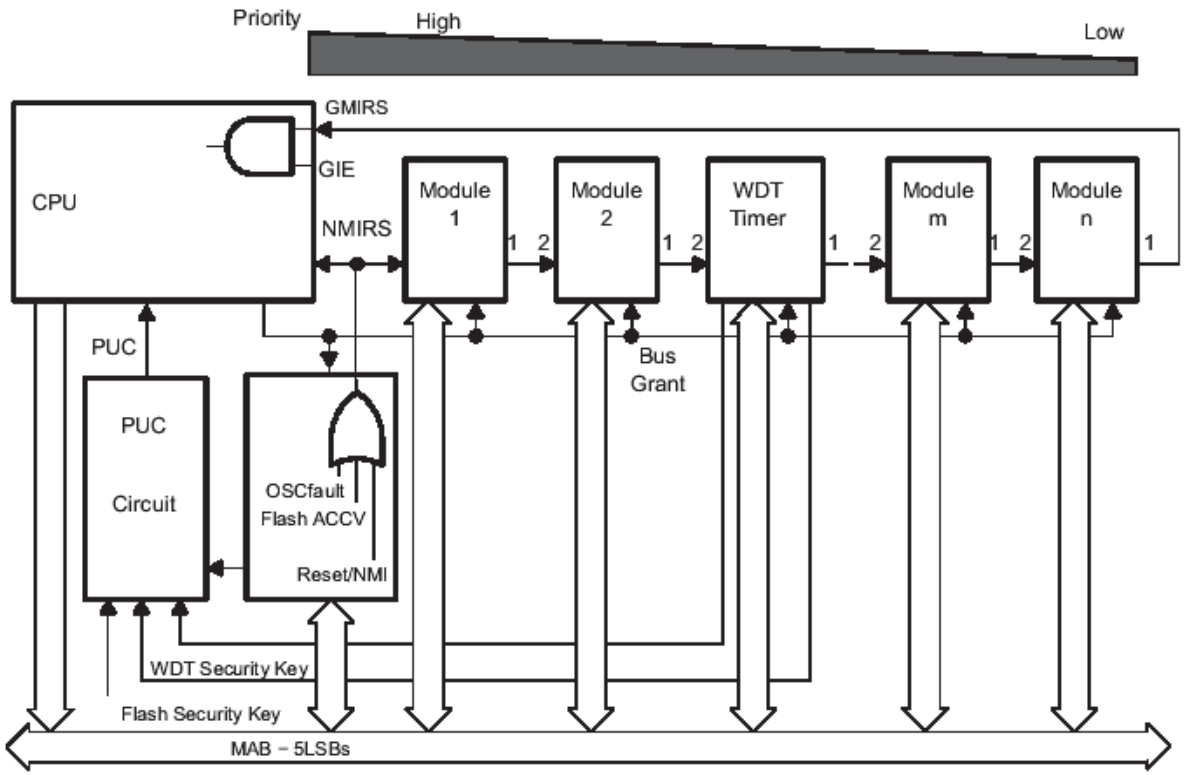
Lorsqu'une interruption survient , les opérations suivantes sont effectuées par le processeur.

1. L'instruction en cours termine son exécution.
2. L'adresse de la prochaine instruction à exécuter est sauvegardée sur la pile.
3. L'adresse de la routine d'exécution appropriée est obtenue en chargeant le contenu d'un vecteur d'interruption.
4. La routine d'interruption est exécutée.
5. A la fin de la routine d'interruption, le processeur reprend l'exécution du programme suspendu à l'adresse sauvegardée précédemment sur la pile.

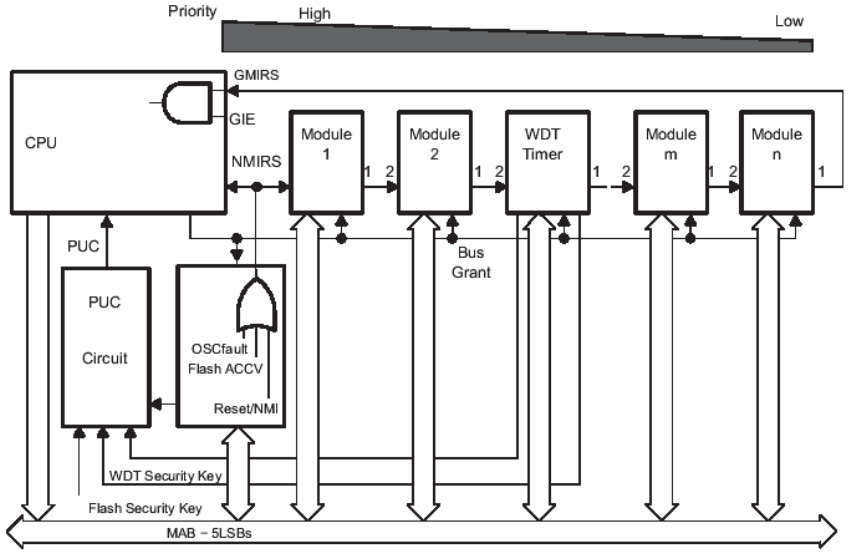
Priorité des interruptions

- Les interruptions sont fixés et définis par la chaine de connexion comme montré dans la figure suivante :

Priorité des interruptions



Priorité des interruptions



- Plus près du CPU un module est, plus haute sera sa priorité.
- Les priorités d'interruptions déterminent quel interruption est choisi lorsqu'il y en a plusieurs simultanés.
- Il y a 3 types d'interruptions :
  - Reset du système
  - Interruption non-masquable (NMI)
  - Interruption masquable

Horloges

ACLK : Auxiliary clock.

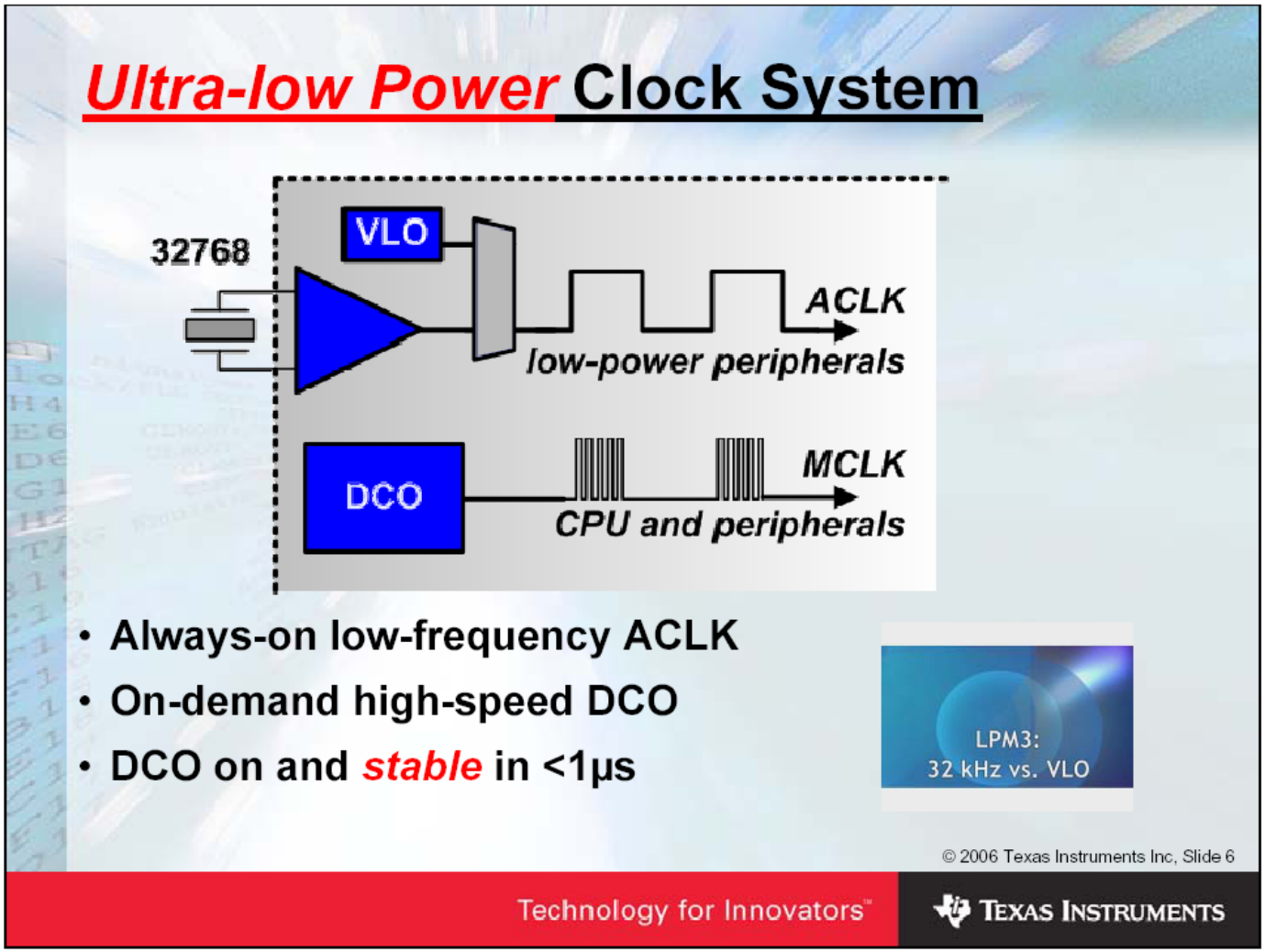
- Horloge basse fréquence (économique) sélectionnable en software par les périphériques.

MCLK : Master clock.

- Horloge haute fréquence, utilisée par le CPU et le système.

SMCLK : Sub-main clock.

- Horloge haute fréquence, sélectionnable en software par les périphériques.



VLO = Very Low power Oscillator
DCO = Digitally Controlled Oscillator

Horloges

- Pour des performances *low-power* optimales, l'horloge **ACLK** peut être configurée pour fonctionner avec un cristal *low-power* (selon les modèles 12 kHz ou 32,768 kHz), fournissant un temps de base stable pour le système et les modes d'opération low-power.
- L'horloge **MCLK** peut être configurée pour fonctionner avec le l'oscillateur on-chip (DCO) et peut être activée seulement lorsque requis par des interruptions.
- L'horloge **SMCLK** (1,049 MHz) peut être configurée pour fonctionner à partir d'un cristal ou du DCO, en fonction des besoins de périphériques.

- Les horloges ont des détecteurs de mauvais fonctionnement et peuvent émettre une interruption dans ce cas.

Entrées- sorties générique (GPIO)

- Les microcontrôleur de la famille MSP430 peuvent avoir jusqu'à 10 ports programmables d'entrée-sortie, **P1** à **P10**.

- Mais les «petits» modèles n'ont souvent qu'un ou deux ports (ex. P1, P2) et quelques broches éparses qui sont accessibles aux broches physiques du circuit.

- Chaque port a **huit broches d'entrée-sortie** et chaque broche est configurable individuellement:
  - programmation / configuration indépendantes pour chaque broche de chaque port,
  - les ports P1 et P2 individuellement configurables comme sources d'interruption,
  - registres d'entrées et registre de sorties séparés et indépendant.

Adressage des entrées-sorties

- Les entrés-sorties physiques (*pins*) sont divisées en *ports.*
- Chaque *port* est contrôlé (lu, écrit, sélectionné, configuré) par une série de registres*:*

General-purpose I/O register address map

		PxIN	PxOUT	PxDIR	PxSEL	PxIES	PxIE	PxIFG	PxREN
P0	0x10	0x11	0x12		0x13	0x14	0x15		
P1	0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27	
P2	0x28	0x29	0x2a	0x2b	0x2c	0x2d	0x2e	0x2f	
P3	0x18	0x19	0x1a	0x1b				0x10	
P4	0x1c	0x1d	0x1e	0x1f				0x11	
P5	0x30	0x31	0x32	0x33				0x12	
P6	0x34	0x35	0x36	0x37				0x13	
P7	0x38	0x3a	0x3c	0x3e					
P8	0x39	0x3b	0x3d	0x3f					
P9	0x08	0x0a	0x0c	0x0e					
P10	0x09	0x0b	0x0d	0x0f					

Adressage des entrées-sorties génériques

- Chaque registre dédié aux entrées/sorties est un registre à 8 bits et est accédé avec des instructions sur un byte.

- Mais les ports P7−P8 et P9−P10 sont arrangées de manière à pouvoir être adressés sous la forme d'un port unique de 16 bits par couple. Dans ce cas le couple P7-P8 désigné sous le nom de la PA et le P9-P10 est adressé sous le nom de PB .
- Par exemple, pour écrire dans les registres (1 byte) de sélection P7SEL et P8SEL simultanément, il est possible de travailler avec un registre de (16 bits), soit PASEL qui est une concaténation des registre P7SEL et P8SEL.

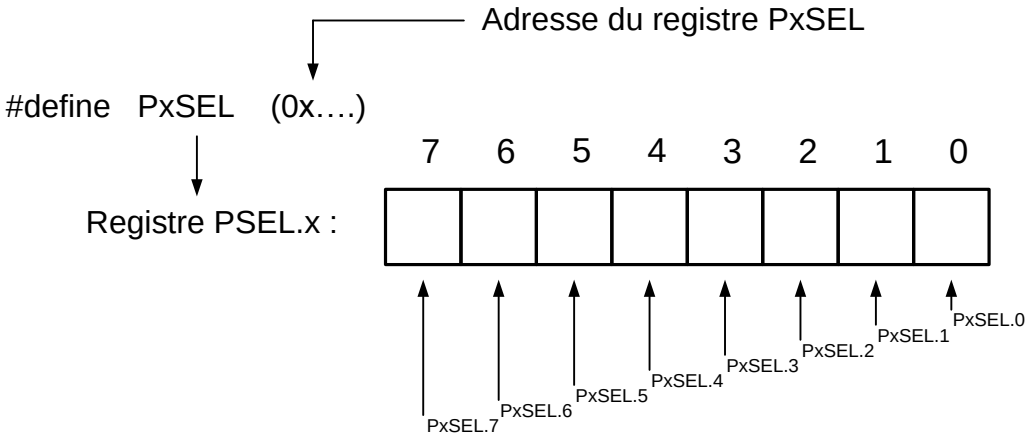
Par exemple en assembleur

```
BIS.B  #0x01,&P7SEL
BIS.B  #0x05,&P8SEL   } BIS.W  #0x0501,&PASEL
```

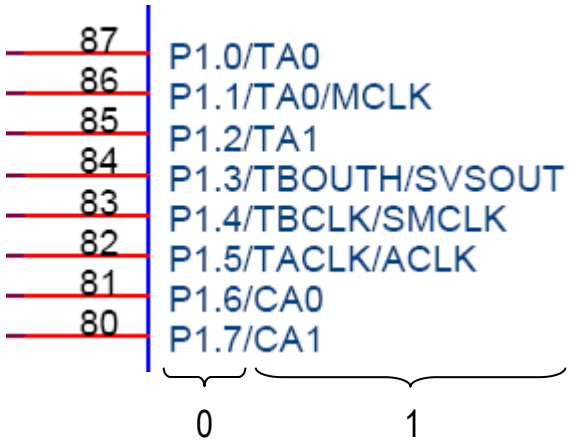
Registre de sélection de fonction : PxSEL

Les **broches physiques** des ports P1 à P10 peuvent être **multiplexées** avec des modules d'autres périphériques. Par conséquent chaque bit des registres PxSEL permet de sélectionner une parmi plusieurs fonctions.

Bit = 0 : la broche est configurée en entrée – sortie
Bit = 1 : la broche est configurée pour une entrée – sortie d'un module spécifique, par exemple une horloge



Exemple : MSP430FG4617
Port 1

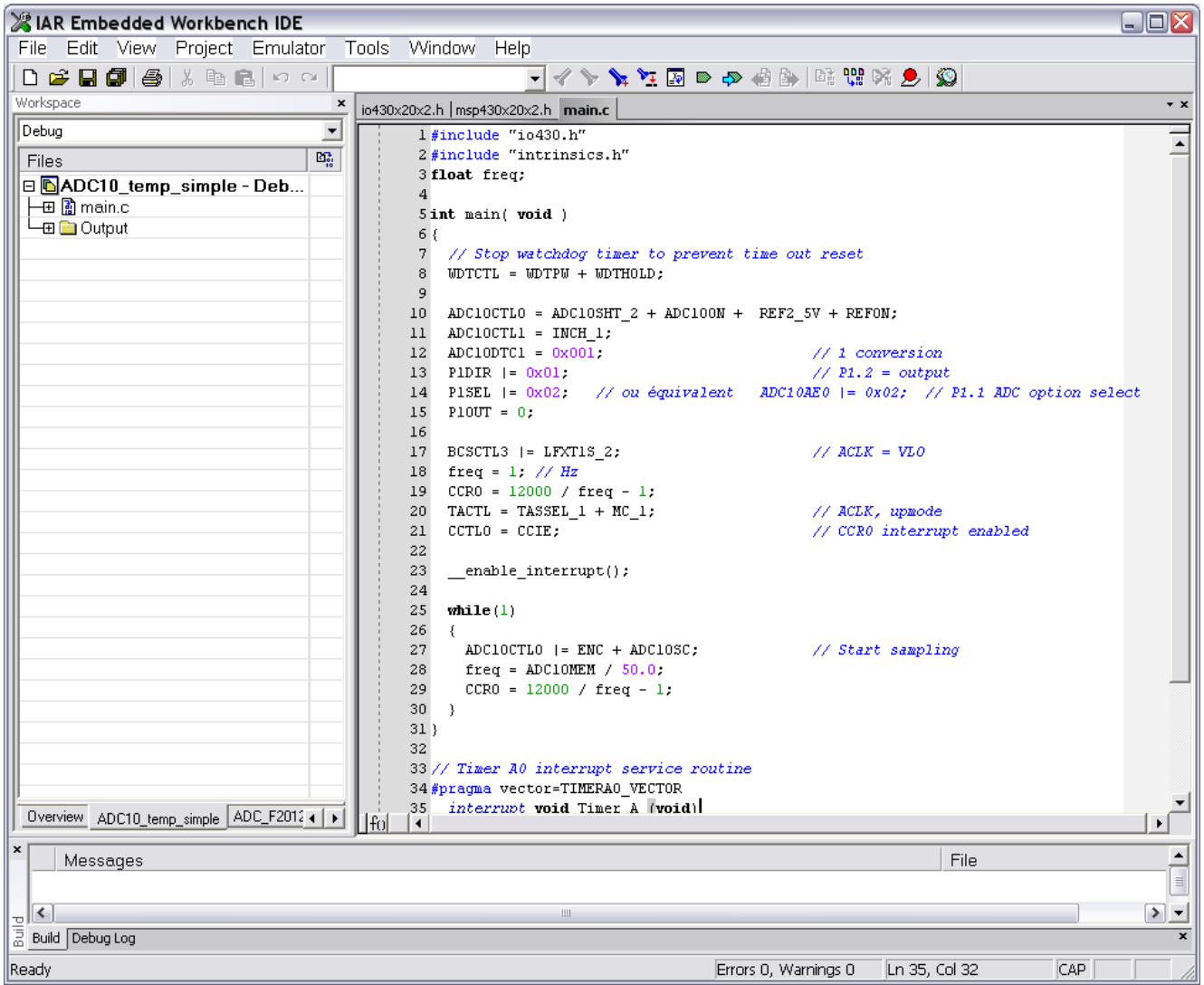


Environnements de développement

- Plusieurs environnements de développement sont disponibles pour ce microcontrôleur.
- ON va utiliser l'environnement IAR Kickstart avec un compilateur IAR C/C++ limité à 4K de code C/C++. Pas de limite pour les programmes en assembleur.
- Le monde du logiciel libre offre aussi une solution MSP-GCC utilisables sur plusieurs plateformes (Windows, Linux, …). Cet environnement est basé sur les compilateur et débogueur GNU.

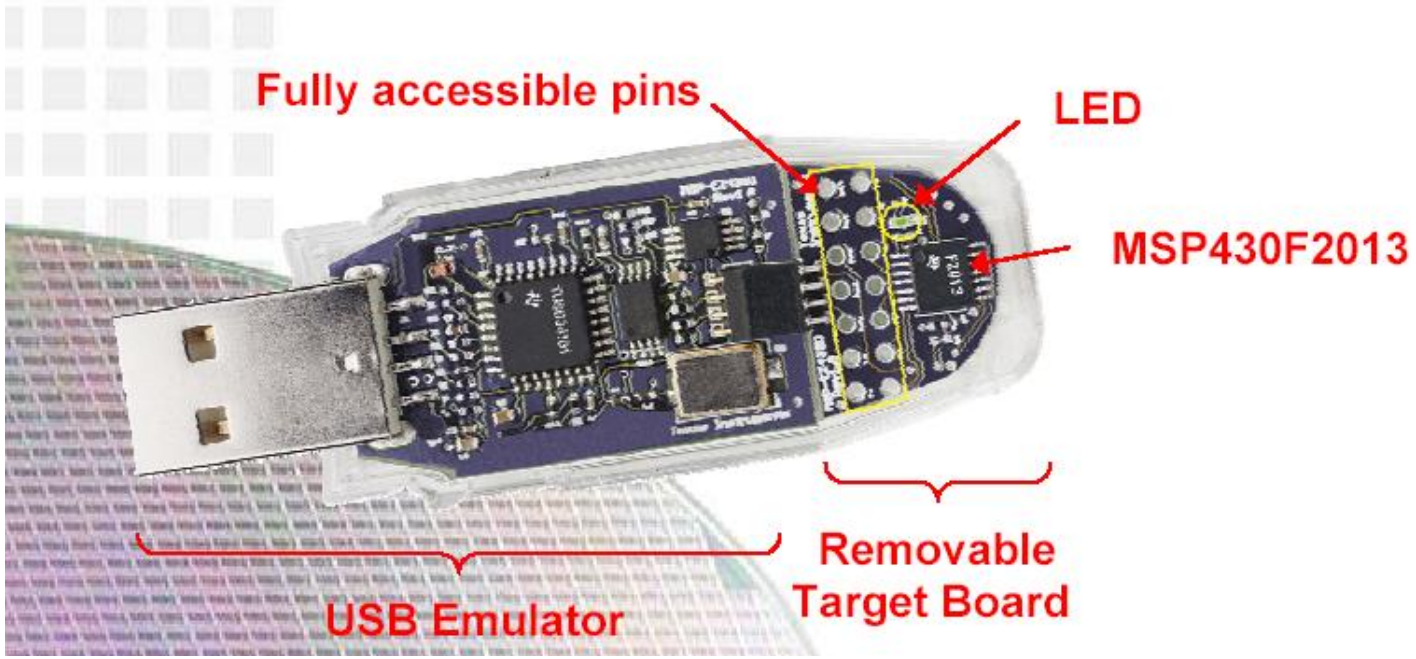
Environnement de programmation

Dans ce cours les programmes du MSP430 seront développés et compilés sur un PC dans l'environnement de développement intégré «IAR Embedded Woorkbench IDE».



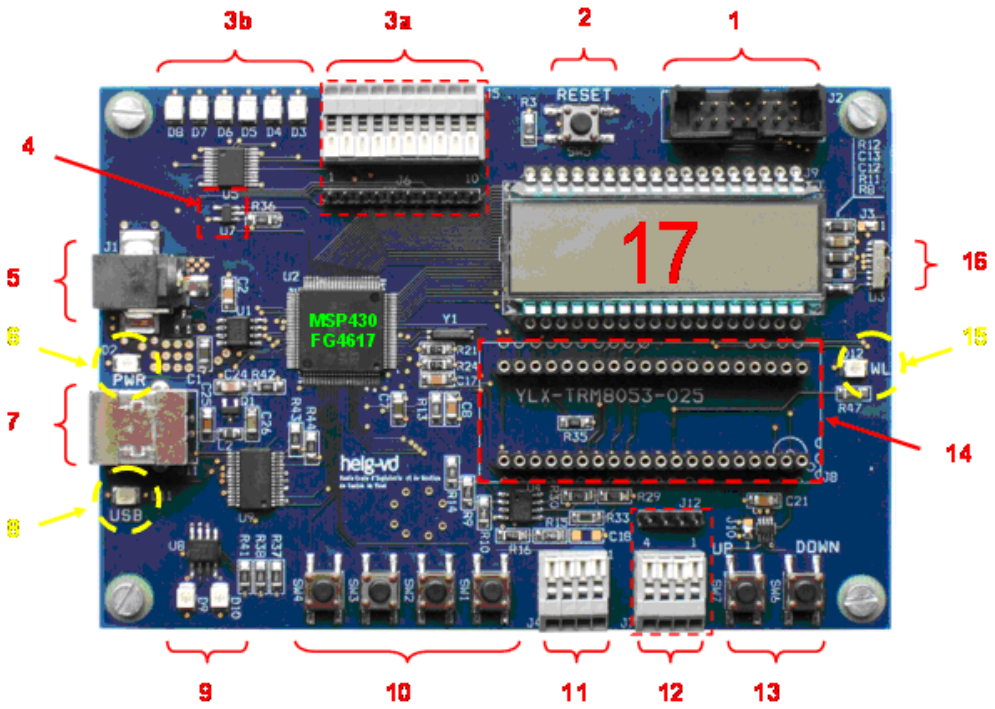
Le eZ430: un outil de développement pour chacun …

- Le MSP430 a aussi un programmateur très économique:
  le eZ430 qui est de la taille d'une clé USB.
- Il sert de programmateur, débogueur et simulateur.
- La petite carte séparable est basée sur un MSP430F012 (ou F013) avec plusieurs entrée/sorties numériques et analogiques.



Pour les applications plus exigeantes: carte de développement du cours MUI

- Une carte de prototypage et d'essai plus élaborée a été **spécialement préparée** pour ce cours.
- Elle inclut un MSP430FG4617 avec des entrées-sorties analogiques et numériques, ainsi que plusieurs périphériques:
  - Interface USB
  - LEDs
  - LCD
  - Capteur de température
  - Module wireless
  - etc.



Vient de sortir chez TI:
MSP-EXP430G2
une carte de développement à 4.30 $/pièce

- Avec interface USB et entrés/sorties accessibles pour toute application.

http://processors.wiki.ti.com/index.php/MSP430_LaunchPad_%28MSP-EXP430G2%29?DCMP=launchpad&HQS=Other+OT+launchpadwiki

Aussi tout récent chez TI:
eZ430-Chronos
une "montre" programmable avec divers capteurs: 49 $

- Basé sur le même module que le eZ430.
- Avec communication sans fil avec le PC, divers capteurs: accéléromètre, pression, température.

eZ430-Chronos
Wireless Development Tool

TEXAS INSTRUMENTS
MSP430

Travail personnel

- Wikipédia en français
  - Microcontrôleur
  - Assembleur
- Wikipédia en anglais
  - MSP430
- *Lire tous ces sujets,*
  noter par écrit ce qui n'est pas (encore) clair, pour des questions et approfondissements successifs en classe.