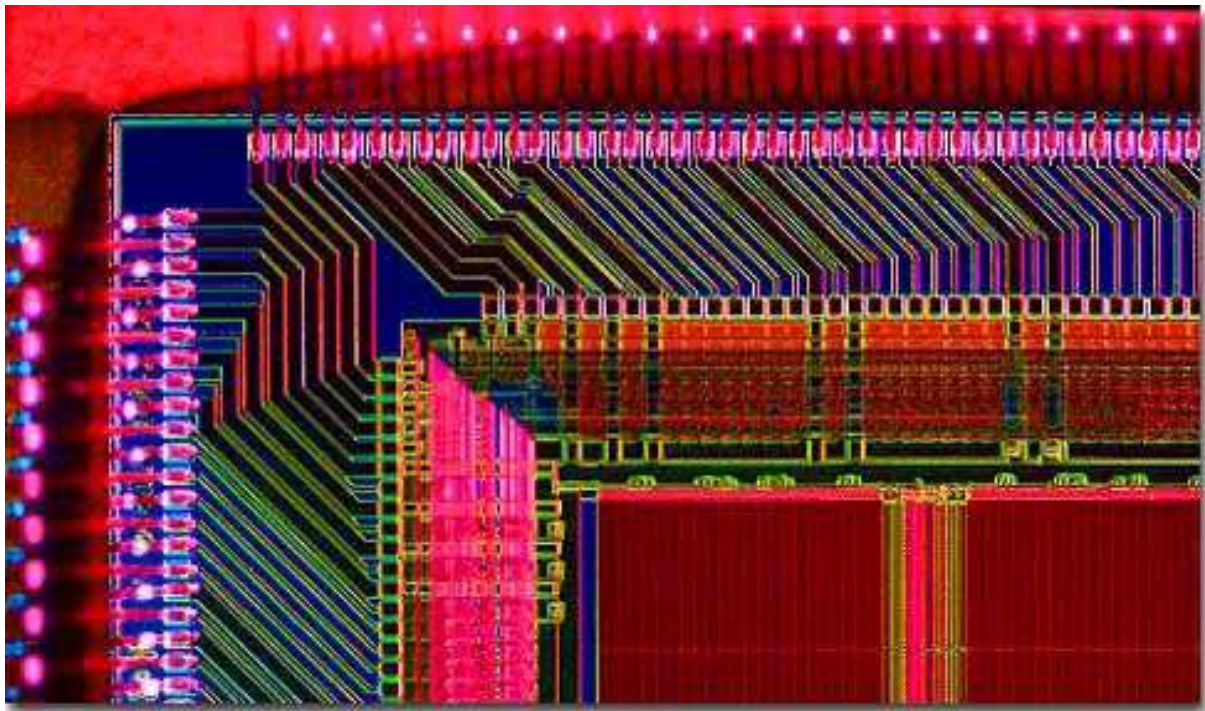



Systemes à Microcontrôleurs

V1.1



 **Ecole Nationale
Supérieure des Mines**
SAINT-ETIENNE
CMP - Georges Charpak

Christian Dupaty
Professeur de génie électrique
Académie d'Aix-Marseille

christian.dupaty@ac-aix-marseille.fr

Table des matières

Table des matières	2
1. Brève histoire du processeur	4
2. Introduction : De l'analogique au numérique	5
2.1. Deux niveaux électriques : le bit	6
2.2. Conventions logiques	6
2.3. Immunité au bruit	6
3. Rappels sur la numération en électronique numérique	7
3.1. Les opérations	7
3.1.1. L'addition	7
3.1.2. Les nombres signés : la soustraction	8
3.2. Les multiplications et les divisions par des puissances de 2	8
3.3. Les opérations booléennes	9
3.3.1. Le complément	9
3.3.2. La fonction « ET » ou « AND »	9
3.3.3. La fonction « OU » ou « OR »	9
3.3.4. La fonction « OU EXCLUSIF »	10
3.3.5. Les fonctions de décalages et de rotations	10
3.4. Les nombres réels	11
3.5. Le codage du texte, la code ASCII	12
4. Introduction au traitement programmé	13
4.1. Algorithme / Organigramme	13
4.2. Structures de programmation	14
4.2.1. Structures alternatives	14
4.2.2. Structures répétitives :	14
4.3. Langages de programmation	15
5. Technologie des systèmes à microprocesseur	16
5.1. Comparatif	16
5.2. Lexique	16
5.3. HARVARD vs VON NEUMAN	17
5.4. MicroProcesseur : Organisation structurelle d'un "système minimum" VON NEUMAN	18
5.5. Exemple de réalisation d'un décodeur d'adresses pour un bus 64KO	19
5.5.1. Décodage par portes logiques	20
5.5.2. Décodage par décodeur intégré :	20
5.5.3. Décodage par PAL :	21
6. Les Microcontrôleurs :	25
6.1. Organisation générale (INFINEON-SIEMENS C517A, CPU type INTEL 8051)	25
6.2. L'unité centrale	26
6.3. Ports parallèles	26
6.4. Ports séries asynchrones	27
6.5. TIMERS	28
6.5.1. Production de signaux, principe :	28
6.5.2. Mesure de durée, principe :	28
7. Le standard INTEL 8051	29
7.1. Une évolution du 8051, l' INFINEON (SIEMENS) C517A	30
7.2. Organisation de la mémoire	31
7.2.1. Unité centrale 8051, éléments d'assembleur	35
7.2.2. Modes d'adressage	35
7.2.3. Le processeur Booleen :	36
8. Le jeu d'instructions du 8051 :	37
9. Programmation structurée et sous-programmes	40
10. L'assembleur 8051	41
10.1. Utilisation du Linker – la définition des segments	42
11. L'environnement de développement	43
12. Interruptions	45
12.1. Principes	45
12.2. Masquage et validation	47
12.3. Exemple de mise en œuvre des interruptions	49
13. TIMER	52
14. Le TIMER 2	57

15.	COMPARE sur C517A.....	58
16.	CAPTURE	61
17.	ADC sur C517A.....	62
18.	Communications séries.....	64

Ressources Internet :

Documentation INTEL 8051 : http://www.intel.com/design/mcs51/cf_51.htm

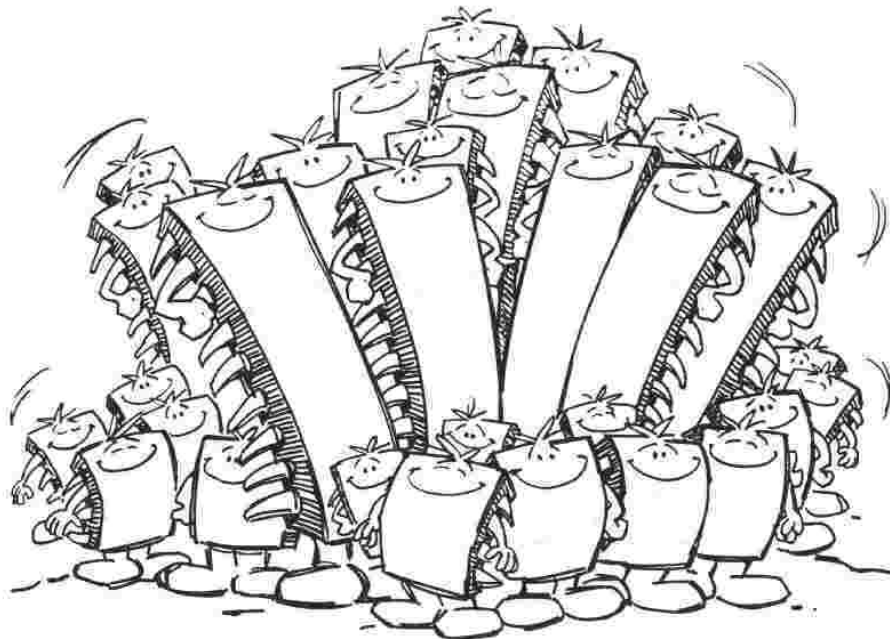
Documentation NXP PHILIPS 8051 : <http://www.nxp.com/>

Documentation ATMEL 8051 : <http://www.atmel.com/products/8051/default.asp>

Outils de développement KEIL : <http://www.keil.com/c51/>

Emulateur/Simulateur 8051 JAVA : <http://www.edsim51.com/>

Emulateur/Simulateur 8051 : <http://www.hugovil.com/fr/emu8051/index.html>



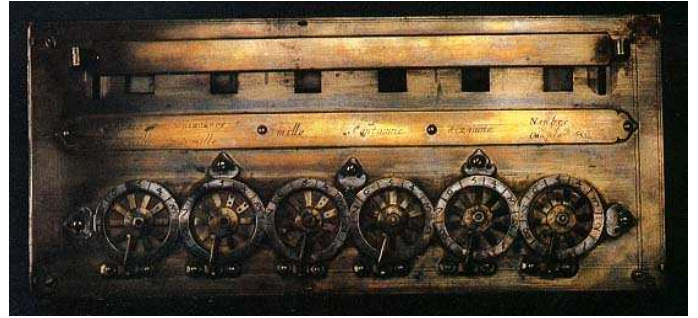
Objectifs et contenus:

- Découverte de la programmation et des microcontrôleurs
- Connaissance de la numération binaire et hexadécimale, fonctions logiques booléennes.
- Codage des nombres entiers des réels et des caractères.
- Structures des systèmes à microprocesseurs, ALU, périphériques, décodage d'adresses.
- Notions d'algorithmie, programmation structurée
- Structures des microcontrôleurs (HAVARD, VON NEUMAN)
- Notions de langage, découverte de l'assembleur
- Structure d'un outil de développement (assembleur, compilateur, linker, simulateur, débbuger)
- Domaines d'emploi et principes des interruptions
- Gestion des périphériques, ports parallèles, TIMER, ports séries, ADC.
- Etre capable créer et valider un petit programme en assembleur en respectant les règles de programmation structurée
- Etre capable de mettre ne œuvre un outil de développement logiciel.

Rq : les exemples de programmes de ce cours sont écrits en assembleur 8051. Les chapitres sur les périphériques prennent exemple sur le 8051 sauf précision (quelques périphériques propres au C517A INFINEON)

1. Brève histoire du processeur

1642 : Pascal invente la Pascaline, calculatrice entièrement mécanique.



La Pascaline

1792 : Les frères Chappe inventent le télégraphe optique (Le premier réseau ?)

1801 : Jacquard : métiers à tisser à cartes perforées

- Langage binaire
- Programme enregistré



Métier à Tisser Jacquard

1854 : Georges BOOLE Autodidacte poussé par De Morgan

1932 : Compteur à tubes (premier « ordinateur » électronique)

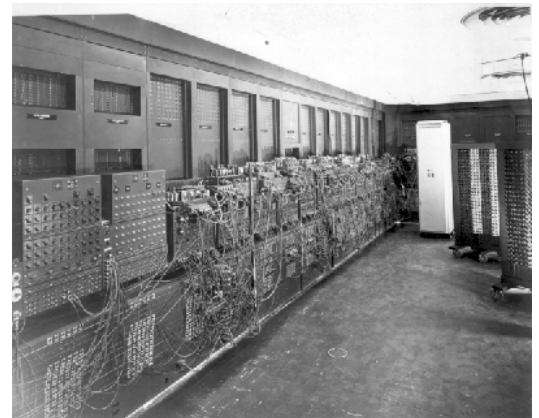
1946 : ENIAC premier ordinateur

Consommation : 140KW

Horloge : 100 kHz

5,000 additions par seconde

500 multiplications par seconde



ENIAC

1951 : Transistor FET et Disque Dur (5Mo)

1953 : Mémoire à Tores magnétiques

1957 : Premier circuit intégré (Texas / Kilby)

1971 : Premier micro-processeur

4004 d'INTEL : 15/11/1971

2250 Transistors Bipolaires

108 KHz, 4bits

1974 : Intel 8080

Mots de 8 bits

bus adresses 16 bits, bus données 8 bits

7 registres 8 bits

64 k octets adressables

6000 transistors

Horloge : 2 MHz

1981 : L'Intel 8086 équipe le premier PC et le MOTOROLA 68000 l'Apple II, début de l'informatique grand public.

1985 1990 : premiers microcontrôleurs industriels INTEL 8051 et MOTOROLA 68HC11.



Premier CI

1995 : La puissance des ordinateurs permet la vulgarisation des outils de CAO et des compilateurs C avec comme cibles les microcontrôleurs

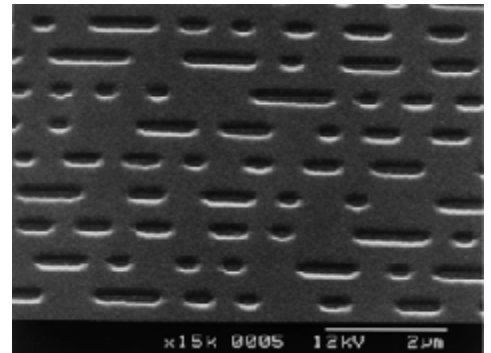
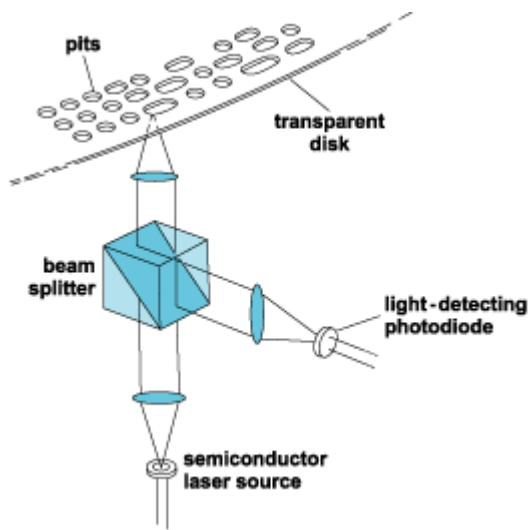
2. Introduction : De l'analogique au numérique

Entre un disque « noir » et un disque « compact » il y a une différence de principe : le premier est *analogique*, le second *numérique*. Que signifient ces termes ?

- Le mot analogique évoque ressemblance, si on regarde avec un microscope une partie de sillon d'un disque noir on verra une sorte de vallée sinuose dont les flancs reproduisent, à peu près, la forme des signaux électriques transmis aux haut-parleurs. A un son grave correspondent des sinuosités qui ont une période spatiale grande (quelques mm pour 100 Hz), à un son aigu correspondent des sinuosités dont la période spatiale est plus petite (quelques centièmes de mm pour 10 kHz). De même, l'amplitude des sinuosités reproduit, grosso-modo, l'amplitude du son que l'on souhaite reproduire.

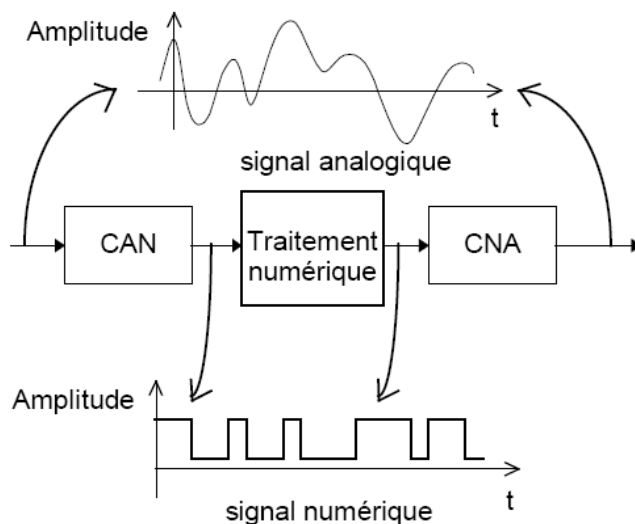
- Le mot numérique évoque nombre. Si on regarde au microscope (Grossissement supérieur à 100) une plage d'un disque compact on verra une sorte de pointillé de trous ovales, presque identiques, répartis de façon irrégulière sur des pistes quasi-circulaires. Aucun rapport de forme entre le son enregistré et l'allure de la gravure ne peut être observé, présence ou absence de trous constituent les deux valeurs possibles d'un chiffre en base 2. Ces chiffres,

regroupés par paquets de 16, constituent des nombres entiers dont la valeur est l'image, via un code, de l'amplitude du signal sonore.



Le passage d'un monde à l'autre se fait par des *convertisseurs analogique numérique* (CAN) et

numérique analogique (CNA), dont nous n'étudierons pas ici le fonctionnement. La différence de principe évoquée plus haut se retrouve évidemment quand on observe le fonctionnement des circuits : un circuit analogique manipule des signaux électriques qui peuvent prendre une infinité de valeurs, qui sont en général les fonctions continues du temps, un circuit numérique manipule des signaux qui ne peuvent prendre qu'un nombre fini (généralement 2) de valeurs conventionnelles, sans rapport avec le contenu de l'information, qui sont des fonctions discontinues du temps.



2.1. Deux niveaux électriques : le bit

Nous considérerons, que les signaux numériques représentent des valeurs binaires ; ils ne peuvent prendre que deux valeurs. Une variable binaire porte le nom de *bit*, contraction de binary digit, littéralement chiffre en base 2. Dans un circuit électronique la grandeur physique significative que l'on utilise le plus souvent est la tension (un signal électrique peut très bien être un courant), sauf précision contraire explicite la « valeur » d'un signal électrique binaire se mesurera donc en volts.

Dans un système numérique tous les potentiels sont mesurés par rapport à un potentiel de référence, la masse, qui est une équipotentielle commune à tous les circuits.

2.2. Conventions logiques

Une entrée ou une sortie d'un circuit numérique ne peut prendre que deux valeurs, notées généralement **H**, pour High (haut), et **L**, pour Low (bas) : 3 et 0,2 volts sont des valeurs typiques fréquemment rencontrées.

La valeur d'un signal représente en général quelque chose :

- chiffre en base deux, 0 ou 1,
- valeur d'une variable logique, vrai ou faux,
- état d'un opérateur, actif ou inactif,
- état d'une porte, ouverte ou fermée,
- état d'un moteur, arrêté ou en marche,
- etc.

Traditionnellement on qualifie de convention logique *positive* l'association entre **H** et 1, ou vrai, ou actif, et de convention logique *négative* l'association entre **H** et 0, ou faux, ou inactif.

2.3. Immunité au bruit

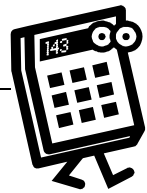
L'un des intérêts majeurs des signaux numériques est leur grande robustesse vis à vis des perturbations extérieures. L'exemple des enregistrements sonores en est une bonne illustration. Les disques noirs reproduisent les altérations mécaniques dues à leur manipulation et à leur vieillissement, le signal sonore s'en retrouve bruité. Le signal numérique ne vieillit pas. Seul son support est altérable.

Au niveau du bit, la protection repose sur le fait que l'information n'est pas contenue dans l'amplitude du signal. Un signal analogique direct a une forme qui est l'image de l'information à transmettre. Toute perturbation à cette forme se traduit par une déformation de l'information associée.

L'amplitude d'un signal numérique n'a pas de rapport avec l'information véhiculée, la seule contrainte est que le système soit encore capable de différencier sans ambiguïté un niveau haut et un niveau bas. L'écart entre ces deux niveaux étant grand, seule une perturbation de grande amplitude pourra provoquer une erreur de décision.

Au niveau du système

Les valeurs élémentaires (bits) sont regroupées en paquets pour former des *mots*, ces mots doivent obéir à certaines règles de construction, des *codes*. On crée des codes qui permettent de détecter et, dans une certaine mesure, de corriger des erreurs.



3. Rappels sur la numération en électronique numérique

En électronique numérique les nombres sont codés en binaire, deux états 0 et 1. Généralement 0 est codé par une tension de 0v et 1 par une tension de 5v. Avec la réduction de la taille des transistors intégrés sur une puce, le 1 est aussi représenté par une tension de 3v voir 1,2v.

Le format de base d'un nombre est l'octet : 8 bits (8 chiffres binaires)

Bit	7	6	5	4	3	2	1	0
Valeur	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Décimale	128	64	32	16	8	4	2	1

Ainsi : 10110110 vaut : $1*2^7+0*2^6+1*2^5+1*2^4+0*2^3+1*2^2+1*2^1+0*2^0 = 128+32+16+4+1= 181$

La notation binaire n'est pas très « lisible » et la notation décimale ne représente pas les bits. Les octets sont généralement codés en hexadécimal.

Ainsi 10110110 se décompose en deux chiffres hexadécimaux B6

1011	0110
B	6

Certains processeurs travaillent sur des nombres de deux octets (16 bits) on parle alors de « mots » (words)

Les processeurs pour ordinateurs les plus modernes travaillent sur des mots de 8 octets (64 bits)

Dec	Hexa
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

3.1. Les opérations

3.1.1.L'addition

Que vaut '1011' + '0110' ?

```

1011
+ 1110
-----

```

- On additionne les chiffres de droite, et on obtient $1+0 = 1$
- On écrit 1
- On additionne $1 + 1$, et on obtient 10 (2 n'existe pas en binaire). On écrit 0 et on reporte 1
- On additionne $0+1$ +le report et on obtient 10. On écrit 0 et on reporte 1
- On additionne $1+1$ +le report et on obtient 11. On écrit 1 et on reporte 1
- Reste le report que l'on écrit, soit 1.

La réponse est donc 11001, soit 25.

Les 2 nombres de départ étant 1011, soit 11, et 1110, soit 14. On procède de la même manière pour les nombres hexadécimaux, en sachant que :

$0xF + 0x1 = 0x10$, soit $15+1 = 16$.

Rq : Les lettres b, h,d désignent la base des nombres , 0x est utilisé en langage C pour désigner un nombre hexadécimal ainsi :

- 10110101b
- B5h
- 0xB5
- 185
- 185d

Désignent tous le même nombre

3.1.2. Les nombres signés : la soustraction

Dans certaines applications, il est nécessaire de pouvoir utiliser des nombres négatifs. Le signe « - » est un concept, il n'est pas représentable électriquement.

Si un nombre binaire est considéré comme « signé » le bit 7 représente le signe (pour un octet). Dans les nombres signés, un bit 7 à '1' signifie nombre négatif.

Remarque : 10000000 (-0) est égal à 00000000 (0). Afin de ne pas « perdre » un nombre la représentation signée est la suivante.

Binaire signé	Dec
00000011	3
00000010	2
00000001	1
00000000	0
11111111	-1
11111110	-2
11111101	-3
11111100	-4
111110101	-5

Pour rendre un nombre négatif

1) On inverse la totalité du nombre.

2) On ajoute 1

On obtient alors ce qu'on appelle le COMPLEMENT A DEUX du nombre.

Exemple : soit le nombre 5 : 00000101 Comment écrire -5 ?

1) on inverse tous les bits (complément à 1) 11111010

2) on ajoute 1 (complément à 2) -5 = 11111011

Pour faire la conversion inverse, on procède de manière identique.

1) on inverse tous les bits 00000100

2) On ajoute 1 00000101

Et on retrouve 5

Dans le cas des octets signés :

La plus grande valeur est 01111111', soit +127

La plus petite valeur devient 10000000', soit -128.

Exemples d'opérations sur les nombres signés

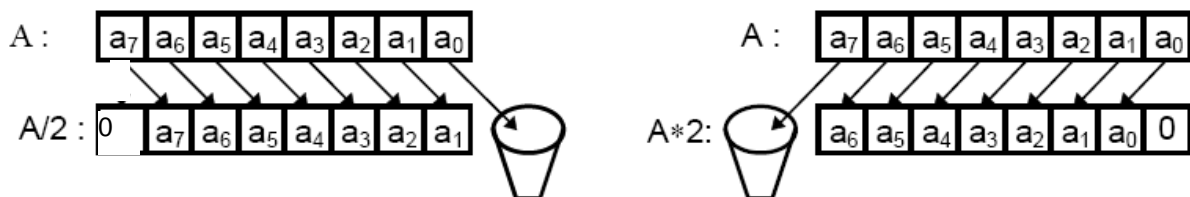
Prenons -3 + 5

$$\begin{array}{r}
 1111101 \text{ (-3)} \\
 + 0000101 \text{ (5)} \\
 = 10000010 \text{ (2)}
 \end{array}$$

Il y a 9 bits, or le processeur n'en gère que 8. Le 9^{ème} est un bit spécial appelé retenue (carry). Dans le registre d'un processeur effectuant cette opération, il reste donc les 8 bits de droite, soit 2, qui est bien égal à (-3) + 5.

3.2. Les multiplications et les divisions par des puissances de 2

Ce sont des décalages arithmétiques (i.e. avec conservation du signe), par exemple pour des octets :





3.3. Les opérations booléennes.

3.3.1. Le complément

Ou « NOT » ou encore complément à 1. Cette fonction est souvent notée « ! »
Elle consiste à inverser tous les bits de l'octet.
Exemple : NOT 10001111 donne 01110000 .

3.3.2. La fonction « ET » ou « AND »

Ou « AND » et souvent notée « & »

Elle consiste à effectuer un « ET » logique entre chaque bits de deux octets. Pour faire une opération « ET », il faut toujours 2 octets.

Les différentes possibilités sont données ci-dessous

Le résultat vaut 1 si et seulement si les deux bits sont à 1

Bit1	Bit2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Exemple :

Soit 11001100' AND B '11110000' donne 11000000'

Cette opération peut être utilisée pour MASQUER des bits et les forcer à 0
Ex pour tester si PA4=1

PORTA	x	x	x	x	x	x	x	x
&	0	0	0	1	0	0	0	0
=	0	0	0	x	0	0	0	0

Le résultat est nul si PA4=0.

Ex positionner PA4 à 0

PORTA	x	x	x	x	x	x	x	x
&	1	1	1	0	1	1	1	1
=	x	x	x	0	x	x	x	x

Seul PA4 a été modifié et mis à 0

3.3.3. La fonction « OU » ou « OR »

Ou OR, et souvent notée « | »

Le résultat vaut 1 si l'un des deux bits est à 1

Bit1	Bit2	OR
0	0	0
0	1	1
1	0	1
1	1	1

Exemple 10001000' OR 11000000' donne 11001000'

Ex positionner PA4 à 1

PORTA	x	x	x	x	x	x	x	x
OU	0	0	0	1	0	0	0	0
=	x	x	x	1	x	x	x	x

Seul PA4 a été modifié et positionné à 1

3.3.4. La fonction « OU EXCLUSIF »

Ou XOR, et souvent notée « ^ »

Le résultat vaut 1 si et seulement si un des deux bits est à 1

Bit1	Bit2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Exemple : 10001000 ' XOR 11000000' donne 01001000 '

Ex Inverser PA4

PORTA	x	x	x	x	x	x	x	x
XOR	0	0	0	1	0	0	0	0
=	x	x	x	/x	x	x	x	x

Seul PA4 a été inversé

3.3.5. Les fonctions de décalages et de rotations

Souvent notées « << » (décalage à gauche) et « >> » décalage à droite

Le contenu de l'octet est décalé d'un bit dans le sens demandé

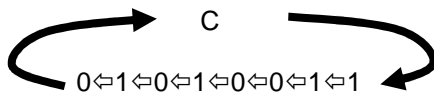
Cas d'un décalage à gauche sur l'octet 01010011:

01010011 << 1 = 10100110

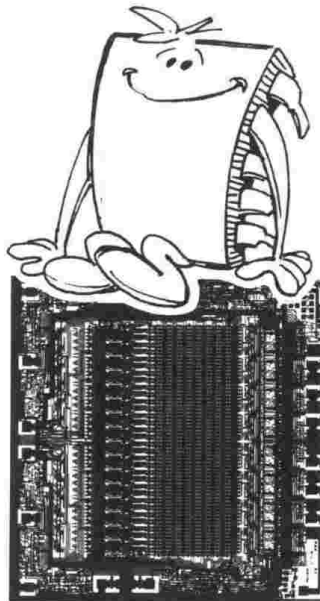
Résultat : 10100110

Remarque 01010011 * 2 = 10100110

Cas d'un rotation à gauche sur l'octet 01010011:



Résultat : 1010011C et C contient 0



3.4. Les nombres réels

Les microcontrôleurs ne peuvent pas effectuer de calculs directs sur les nombres réels, ceux-ci ne sont pas codables en binaire (il n'y a que des nombres entiers). Les nombres réels sont codés en virgule fixe ou en virgule flottante, ils ne sont pas utilisables pour les programmes écrits en assembleur.¹

Calcul en virgule fixe, Le calcul s'effectue en codage Q

On choisit (selon les besoins) un nombre de bits pour la partie entière et un nombre de bits pour la partie décimale. Les bits de la partie entière représentent les puissances de 2^n et ceux de la partie décimale les 2^{-n} . L'erreur absolue sur un réel représenté en virgule fixe est toujours inférieure à 2^{-m}

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
256	128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125	0.00390625

Exemple : codage de la valeur 1,5625 sur un octet positif en Q6

6 chiffres binaires décimaux

1 chiffre binaire pour la partie entière (0 ou 1 donc)

le poids fort représente le signe (0 pour +, 1 pour -)

Partie décimale (ici 6 bits en Q6)

$$1,5625 = 0110\ 0100 = 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 0 \cdot 2^{-5} + 0 \cdot 2^{-6} = 1 + 0.5 + 0.0625 = 1.5625$$

Signe +

Partie entière (ici 1 bit)

-1,5625 = 11011100 en complément à 2

Le complément à 2 de la partie entière 01 donne 10+1=11 (complément +1)

Le complément à 2 de la partie décimale 100100 donne 011011+1=011100

Calcul en virgule flottante

Ce type de codage est très performant car il permet de coder les nombres réels sur une très grande plage numérique avec un format binaire constant. Il demande en revanche une manipulation binaire plus complexe pour les additions et les multiplications et généralement utilisé par les langages évolués comme le C.

$$r = S_m.M.10^{S_e.E}$$

r : réel à coder

S_m : signe de la matrice (0 = positif, 1 = négatif)

M : matrice

S_e : signe de l'exposant

E : exposant

Un nombre réel codé en virgule flottante a cet aspect :



Nb bits	Exposant N	Mantisse M
16	4	10
32	8	22
64	14	48

16, 32, 64 ou 128 bits suivant les processeurs et les compilateurs

La mantisse représente les chiffres significatifs d'un réel inférieur à 0 codé en 2^{-n}

Par exemple 245,36 a une mantisse égale à +24536 et un exposant égale à +3 : $245,36 = 0.24536 \cdot 10^3$

Exemple de codage en virgule flottante :

$$-5,635 = -0,5635 \cdot 10^1 \quad \text{et} \quad 2^{-1} + 2^{-4} + 2^{-10} = 0.5634765$$

S_m	S_e	E_3	E_2	E_1	E_0	M_9	M_8	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0
1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	1

1

0.5634765

¹ Les compilateurs C ou C++ s'appuient sur des bibliothèques écrites en assembleur et effectuant les quatre opérations sur les réels. Les fonctions mathématiques complexes sont réalisées par développements limités avec ces quatre opérations

3.5. Le codage du texte, la code ASCII

Dans les années 60, le code ASCII (American Standard Code for Information Interchange) est adopté comme standard pour les télécriteurs. Il permet le codage de caractères sur 8 bits, soit 256 caractères possibles, normalisé de 0x00 à 0x7F (bit de poids fort à 0) il est adapté aux particularités linguistiques telles les accents pour les codes 0x08 à 0xFF. Il est resté de fait le standard de communication en mode texte.

Le code ASCII

code		1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	NP	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL



4. Introduction au traitement programmé

4.1. Algorithme / Organigramme

Représentation graphique normalisée d'un processus.

Ensemble de règles opératoires rigoureuses permettant de décrire un processus particulier.

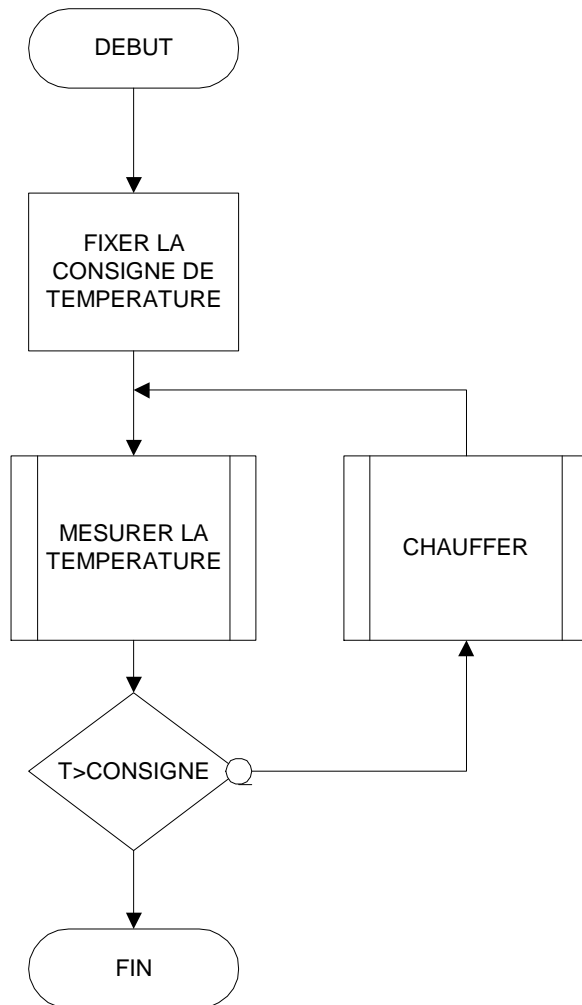
Ex. : algorithme de mesure, de conditionnement de bouteilles, de régulation d'un réacteur nucléaire...

Un algorithme peut être :

- représenté par un **organigramme**.
- écrit sous forme littérale, avec un **langage algorithmique** proche d'une programmation structurée type C.

Exemple :

Algorithme



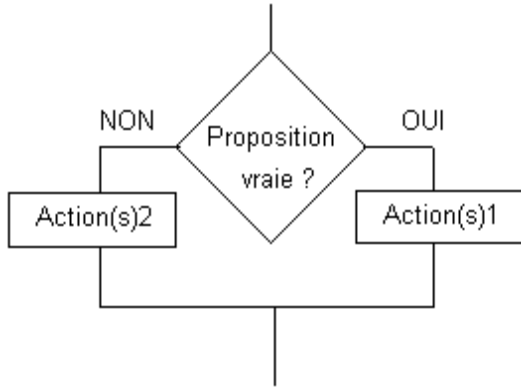
Algorithme

```

Debut
  Fixer la température de consigne
  Tant que T < Consigne
    Chauffer
  Fin Tant que
Fin
  
```

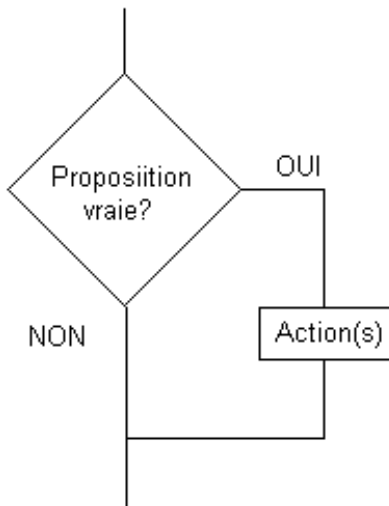
4.2. Structures de programmation

4.2.1. Structures alternatives



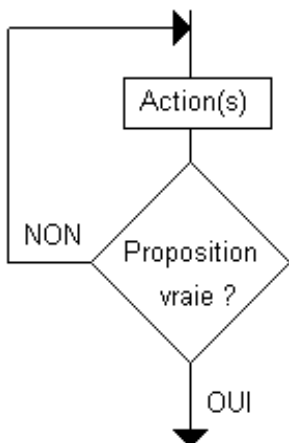
SI proposition vraie **alors** action(s)1
sinon action(s)2
Fin si

ou sous forme réduite :

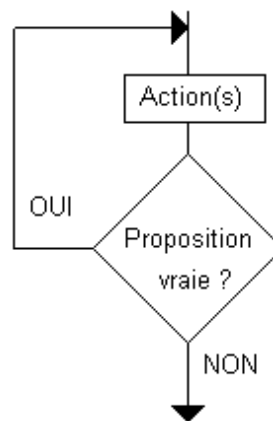


SI proposition vraie
alors action(s)
Fin si

4.2.2. Structures répétitives :



Répéter action(s)
proposition vraie



jusqu'à

Répéter action(s)
tant que proposition vraie

4.3. Langages de programmation

L'exécution d'un PROGRAMME se définit comme la lecture, le décodage et l'exécution d'une suite d'instructions stockées en mémoire.

Langage machine :

Le μP est constitué de circuits logiques séquentiels et combinatoires. La programmation de ces circuits dépend des niveaux logiques présents dans le *registre d'instruction*. Pour être stocké en mémoire le programme doit être écrit en binaire. A la base, les instructions sont donc codées en *langage binaire* (hexa) : ce sont les instructions machines.

Chaque μP possède son jeu d'instructions. Devant l'impossibilité d'écrire un programme en langage binaire on utilise des **logiciels de programmation** permettant la conception de programmes dans un langage plus ou moins proche de l'homme.

Exemple de CODE MACHINE (μC 8051) :

0x0000	75817F		
0x0003	B2A0		
0x0005	1109		Illisible !!!
0x0007	80FA		
0x0009	78FF		
0x000B	79FF		
0x000D	D9FE		
0x000F	DBFA		
0x0011	22		

Programmation en ASSEMBLEUR :

Chaque instruction machine est représentée par une écriture symbolique (mnémonique) : le langage assembleur qui dépend du μP utilisé (pas de compatibilité). La programmation ASSEMBLEUR est essentiellement :

Une traduction du langage mnémonique en langage machine. (Assembleur)

Une affectation des valeurs numériques aux étiquettes et opérandes (éditeur de liens ou LINKER)

Exemple de programme assembleur (μC 8051) :

```
; FLASH LED PORT50
LED      EQU      P5.0
          CSEG    AT 0
          MOV     SP,#7Fh ; initialize stack
SUITE:   CPL     LED      ; flash LED
          CALL   TEMPO
          SJMP   SUITE
;
TEMPO:   MOV     R0,#0FFh      ; tempo longue (R0xR1)
TEMPO1:  MOV     R1,#0FFh
TEMPO2:  DJNZ   R1,TEMPO2
          DJNZ   R0,TEMPO1
          RET
          END
```

Programmation en langage évolué :

La programmation en langage assembleur dépend du μP utilisé. Elle est efficace mais peu lisible et difficile à maintenir.

Un langage évolué utilise des instructions très synthétiques. La traduction est effectuée par un programme COMPILATEUR dont le rôle englobe celui de l'assembleur. Le rôle du compilateur est de traduire un programme SOURCE écrit en langage évolué en un programme OBJET écrit en langage machine directement exécutable par le μP .

Exemple de programme en langage C : (partiel)

```
char mi, ce, di, un; /* définition des variables utilisées */
int i,m, c, d, u;
PORT=124 ;
if (bouton==enfonce) Led=eteind ;
else Led= allume ;
m=valeur/1000; /* Valeur du chiffre du millier */
c=(valeur-(1000*m))/100; /* Valeur du chiffre de la centaine */
```

La programmation en langage évolué **est indépendante du μP utilisé.**

5. Technologie des systèmes à microprocesseur

5.1. Comparatif

	Petits automatismes	Informatique industrielle
Fréquences	4 à 40MHz	1GHz à 3.5GHz
Puissance de calcul	faible	Très élevé
Mémoire	RAM ² : 1KO ³	RAM : 1GO ROM ⁴ : 1MO
Mémoire programme	ROM FLASH ⁵ : 64 KO	Disque dur
Intégration des périphériques	Maximum	minimum
consommation	faible	Elevé
prix	0,5€ à 50€.	200€ à 1000€
Composant utilisé	Microcontrôleur 8/16 bits	Microprocesseur 32/64 bits

Note : les structures des systèmes pour l'informatique industrielle ne sont pas présentées ici

5.2. Lexique

- **UP** : 8 bits, 64KO d'adresses, type 8051, 68HC11, PIC16, PIC18, ATMEL AVR, ST6, ST7...
- **ROM** : mémoire morte PROM ou EPROM
- **RAM** : Mémoire vive statique
- **PIA, PIO** : Peripheral interface adapter, périphérique input/output. Port parallèle 8bits
- **ACIA ou USART** : Asynchronous communications interface adapter ou Universal Synchronise/Asynchrone Receive/Transiver. Port série code NRZ.
- **TIMER** : PTM : Programmable Timer Module. Production et mesures de durées.
- **Décodeur d'adresses** : permet à partir du bus d'adresse (A0-A15) ou d'une partie de celui ci de créer des signaux permettant de sélectionner les différents boîtiers (CS pour Chip Select) lorsque l'adresse du boîtier considéré est active.

De nombreux autres périphériques existent sur le marché:

- EEPROM : mémoire morte effaçable électriquement
- ADC : convertisseurs analogiques numériques
- DAC : convertisseurs numériques analogiques
- Compérateurs de tension
- Horloge temps réel (RTC)
- Port parallèle de puissance
- Commande de moteur pas à pas ou PWM (gestion de pont en H)
- Interface série synchrone SPI ou IIC
- Gestionnaire afficheur LCD graphique / alphanumérique
- Interface ethernet
- Interface WIFI
- Bus de communications de terrain (CAN ou LIN)
- Superviseur d'alimentation / Chien de garde (watch dog)

Principaux fondateurs de microcontrôleurs 8bits(2008) :

- ATMEL (AVR)
- INFINEON
- INTEL (805x)
- MICROCHIP (PIC)
- MOTOROLA (68HCxx)
- NEC
- STMicroelectronics (STx)
- TOSHIBA

² Ram Access Memory : Mémoire « vive », en lecture/écriture elle ne conserve les informations que si elle est sous tension

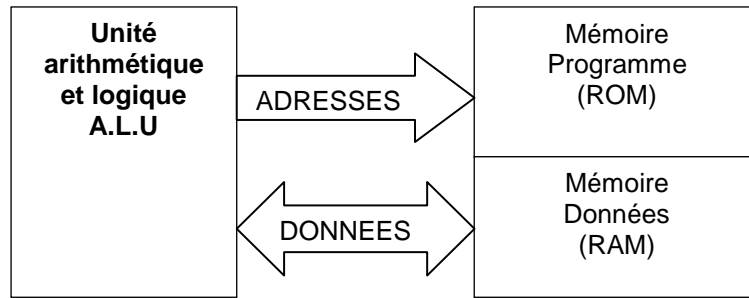
³ KO : Kilo Octets, 1 KO = 2¹⁰ octets = 1024 octets, MO Mega Octets, 1 MO = 2²⁰ octets = 1.048.576 octets

⁴ Read Only Memory : Mémoire « morte », elle conserve ses données même hors tension, on ne peut pas écrire dedans

⁵ Mémoire ROM effaçable électriquement et réinscriptible (environ 100.000 fois) le temps d'écriture est 10⁶ fois supérieur à celui d'une RAM

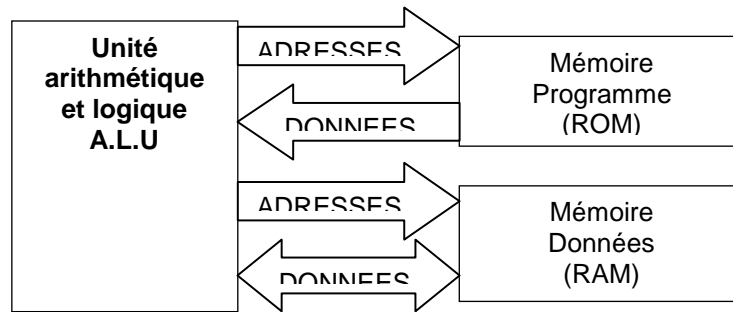
5.3. HARVARD vs VON NEUMAN

- L'architecture, dite architecture de **Von Neumann**, est un modèle pour microcontrôleur qui utilise une structure de stockage unique pour conserver à la fois les instructions et les données requises ou générées par le calcul. La séparation entre le stockage et le processeur est implicite dans ce modèle. Une même instruction permet l'accès au programme ou aux données, cependant pas simultanément. Cette architecture est maintenant principalement utilisée pour la conception des processeurs d'ordinateurs (PC, MAC)



Ce type d'architecture nécessite plusieurs cycles d'horloge pour exécuter une instruction (recherche d'instruction, décodage, lecture opérande, exécution)

- L'architecture de type **Harvard** est une conception de [microprocesseurs](#) qui sépare physiquement la mémoire de données et la mémoire programme. L'accès à chacune des deux mémoires s'effectue via deux bus distincts. Cette structure permet un accès simultané aux données et aux instructions l'exécution des programmes est plus rapide. En revanche elle nécessite des instructions différentes pour accéder à la mémoire programme et à la mémoire de données. Cette architecture très employée pour la conception des processeurs de traitement de signal (DSP) est de plus en plus utilisée pour les microcontrôleurs d'usage généraux.



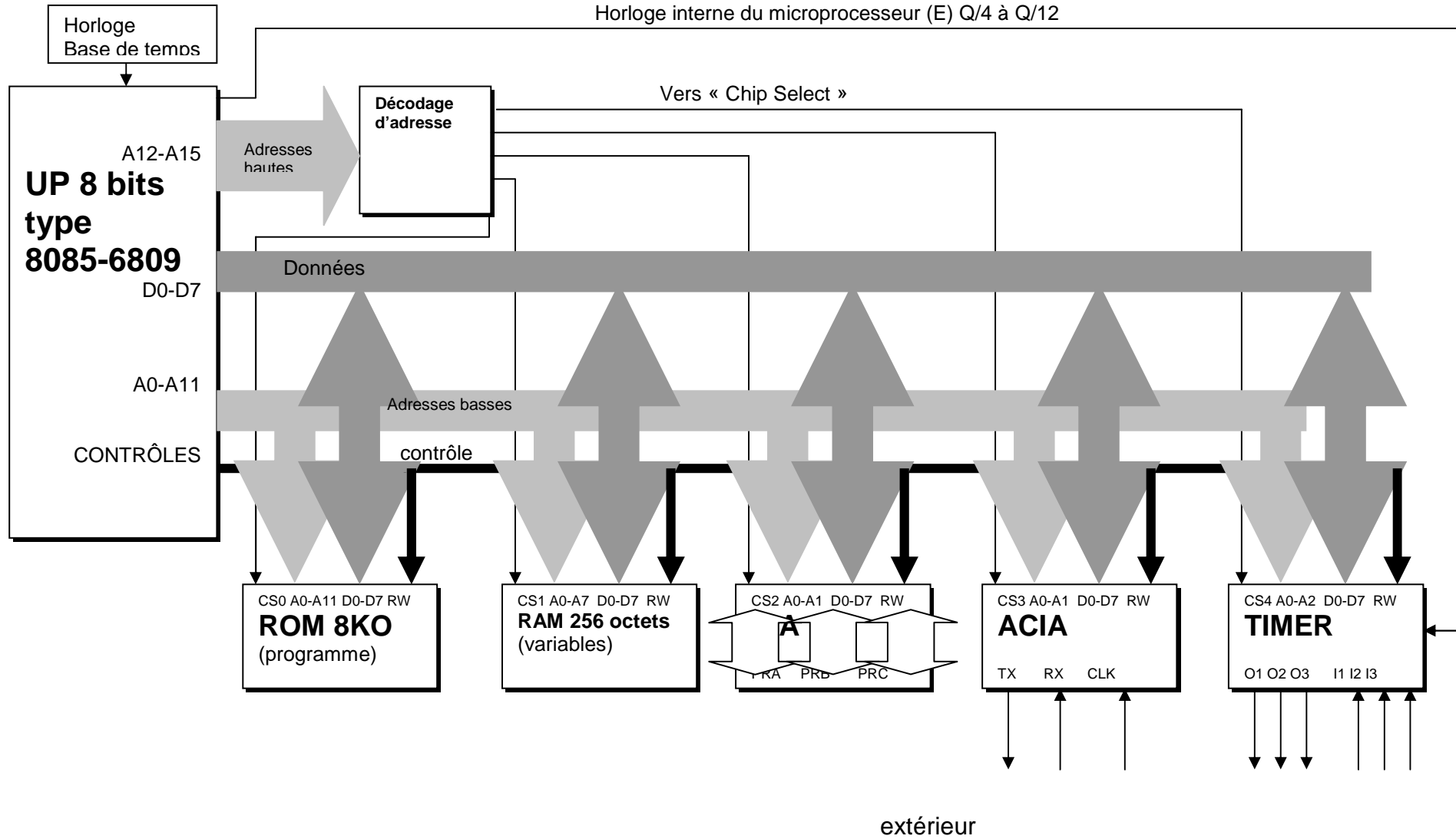
Ce type d'architecture, plus complexe à fabriquer permet une exécution du programme en PIPELINE (recherche d'instruction, décodage, lecture opérande, exécution en un cycle d'horloge)

Instruction/temps	T	T+1	T+2	T+3	T+4	T+5	T+6
N	R	D	L	E	R	D	L
N+1		R	D	L	E	R	D
N+2			R	D	L	E	R
N+3				R	D	L	E

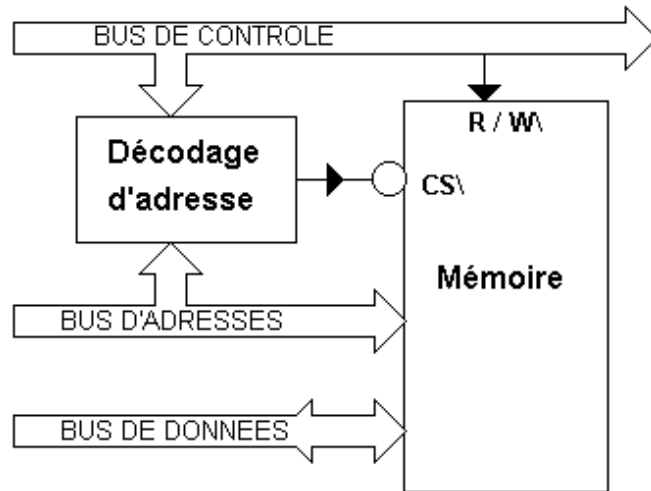
R : recherche
D : Décode
L : Lit opérande
E : Exécute

A partir de ce temps le microcontrôleur effectue les tâches RDLE en un cycle d'horloge

5.4. MicroProcesseur : Organisation structurale d'un "système minimum" VON NEUMAN



LE DECODAGE D'ADRESSES :



Le décodage d'adresse permet de définir l'emplacement des différentes zones mémoires dans l'espace adressable. Pour cela sa fonction est de produire les différents signaux de sélection (CS\') des circuits communiquant avec le µP : mémoires, CNA, CAN, circuit d'interface //, série ...

Si un circuit n'est pas sélectionné (CS\'=1), ses entrées/sorties se trouvent en état HAUTE IMPEDANCE avec pour effet de DECONNECTER le circuit du bus de données.

5.5. Exemple de réalisation d'un décodeur d'adresses pour un bus 64KO

\$FFFF \$F000	PROM1 4KO
\$EFFE \$E000	PROM2 4KO
\$A007 \$A000	TIMER
\$6004 \$6000	ACIA
\$4004 \$4000	PIA
\$07FF \$0000	RAM 2KO

type 68xx , sur un microcontrôleur INTEL 80xx la PROM commence à l'adresse 0x000

Table de vérité (* : n'est pas nécessaire au décodage)

32K	16K	8K	4K	2K	1K	512	256	128	64	32	16	8	4	2	1	Hexadécimal	/CS
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0		
1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	\$F000-\$FFFF	ROM1
1	1	1	0	x	x	x	x	x	x	x	x	x	x	x	x	\$E000-\$FFFF	ROM2
1	0	1	*	*	*	*	*	*	*	*	*	*	x	x	x	\$A000-\$A007	TIMER
0	1	1	*	*	*	*	*	*	*	*	*	*	*	x	x	\$6000-\$6003	ACIA
0	1	0	*	*	*	*	*	*	*	*	*	*	*	x	x	\$4000-\$4003	PIA
0	0	0	*	*	x	x	x	x	x	x	x	x	x	x	x	\$0000-\$07FF	RAM

Remarque : il existe des adresses « images », c'est à dire des adresses où l'on retrouvera la même donnée.

Ex : la donnée à l'adresse \$0080 de la RAM se retrouvera aux adresses \$0080, et \$0880 les bits d'adresse A8-A12 n'étant pas décodés.

Le décodage d'adresse peut s'effectuer de diverses manières : par portes logiques, décodeur intégré (type 74LS138) ou par PAL (type 16L8), il s'agit de toutes les façons de logique combinatoire.

5.5.1. Décodage par portes logiques

Les /CS des circuits périphériques et mémoires sont généralement actifs à « 0 », on obtient donc les équations suivantes :

$$\text{RAM} = \overline{A15} + \overline{A14} + \overline{A13} + \overline{A12}$$

$$\text{PIA} = \overline{A15} + \overline{A14} + \overline{A13} + A12$$

$$\text{ACIA} = \overline{A15} + \overline{A14} + \overline{A13} + A12$$

$$\text{TIMER} = \overline{A15} + \overline{A14} + \overline{A13} + A12$$

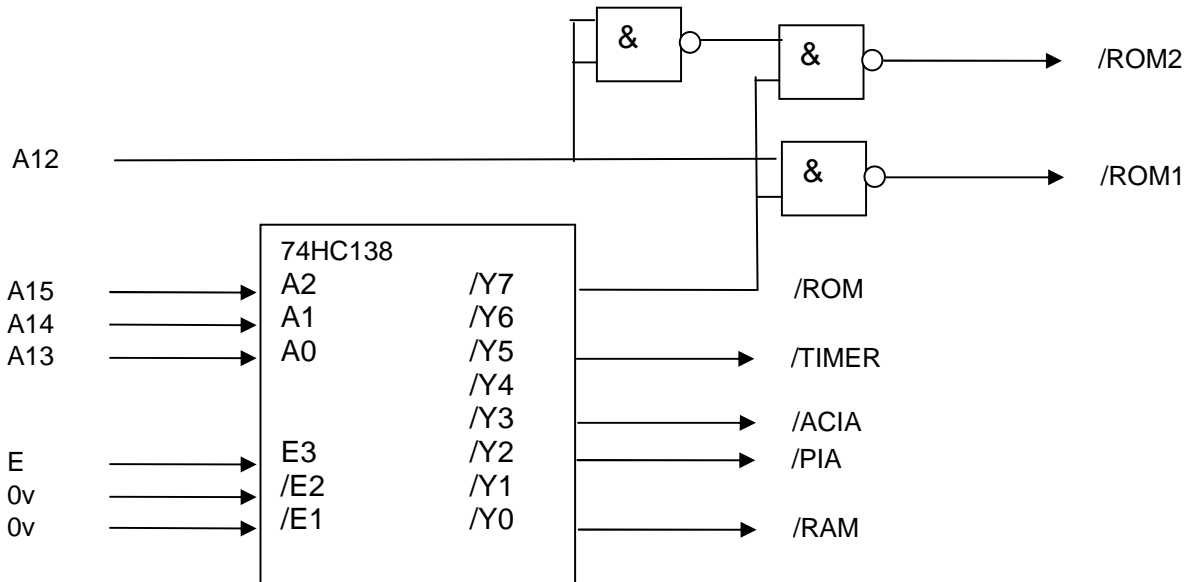
$$\text{ROM2} = \overline{A15} + \overline{A14} + \overline{A13} + A12$$

$$\text{ROM1} = \overline{A15} + \overline{A14} + \overline{A13} + \overline{A12}$$

On voit que la réalisation de ce décodeur nécessitera plusieurs circuits intégrés (des fonctions NAND, NOR, AND, OR), de plus certains périphériques nécessitent d'être synchronisés avec l'horloge E du microprocesseur, les signaux /CS doivent alors être validés lors de l'état actif de E ce qui nécessite des portes supplémentaires. Cette solution est très rarement retenue.

5.5.2. Décodage par décodeur intégré :

le 74HC138 est un décodeur 3 vers 8 avec 3 entrées de validation et sorties actives à l'état bas.



Le décodage est ici simplifié, cependant trois portes NAND sont encore nécessaires.

En plaçant /ROM1 sur /Y7 et /ROM2 sur /Y6 on élimine le circuit NAND mais le décodage d'adresse est modifié :

/ROM1 va de \$F000 à \$FFFF mais avec des images de \$E000 à \$EFFF !

/ROM2 va de \$D000 à \$DFFF mais avec des images de \$C000 à \$CFFF !

5.5.3. Décodage par PAL :

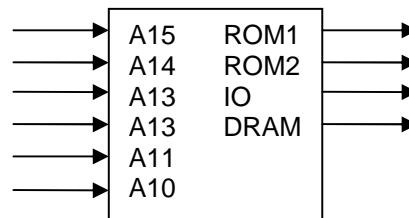
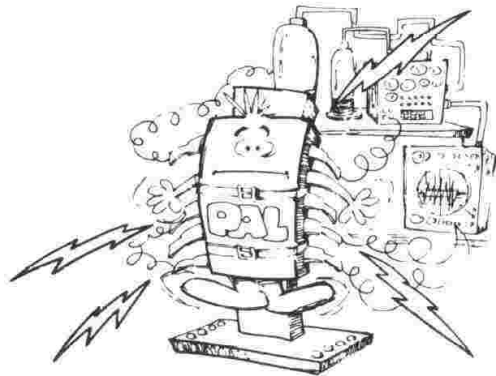
L'utilisation d'un PAL (programmable array logic) permet un découpage beaucoup plus souple pour un prix similaire du composant, cependant la programmation d'un PAL nécessite un programmeur et un outil de développement de PAL en langage ABEL ou VHDL .

Exemple de décodeur d'adresse écrit en langage ABEL

```

module DECODAD;
title 'Decodeur d adresse';
Declarations
    DECODAD device 'P16L8';
    A15,A14,A13,A12,A11,A10 pin 1,2,3,4,5,6;
    ROM1,IO,ROM2,DRAM pin 14,15,16,17;

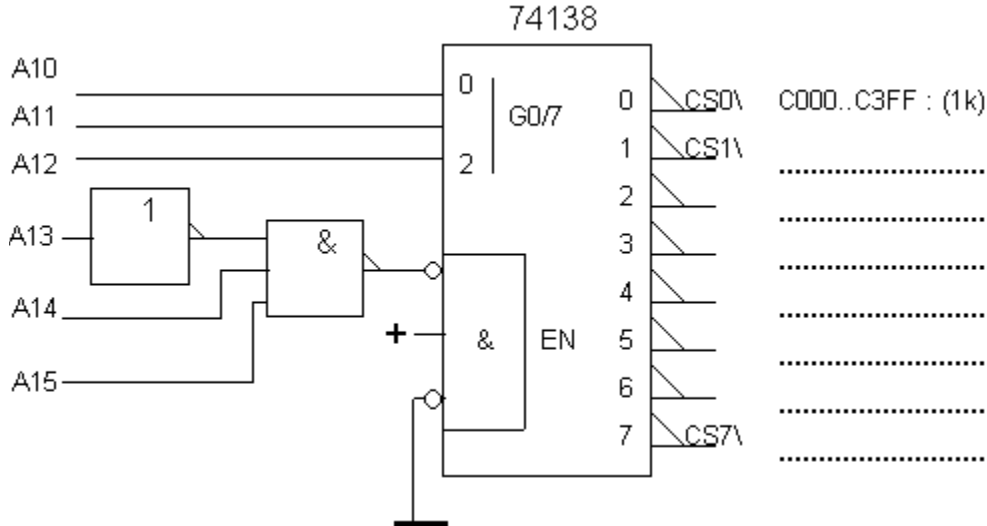
    H,L,X = 1,0,.X.;
    Address = [A15,A14,A13,A12, A11,A10,X,X, X,X,X,X, X,X,X,X];
equations
    !DRAM = (Address <= ^hDFFF);
    !IO = (Address >= ^hE000) & (Address <= ^hE7FF);
    !ROM2 = (Address >= ^hF000) & (Address <= ^hF7FF);
    !ROM1 = (Address >= ^hF800);
test_vectors
    (Address -> [ROM1,ROM2,IO,DRAM])
    ^h0000 -> [ H, H, H, L ];
    ^h4000 -> [ H, H, H, L ];
    ^h8000 -> [ H, H, H, L ];
    ^hC000 -> [ H, H, H, L ];
    ^hE000 -> [ H, H, L, H ];
    ^hE800 -> [ H, H, H, H ];
    ^hF000 -> [ H, L, H, H ];
    ^hF800 -> [ L, H, H, H ];
end DECODAD
  
```



Exercice : Décodage par décodeur 3/8 (74138) :

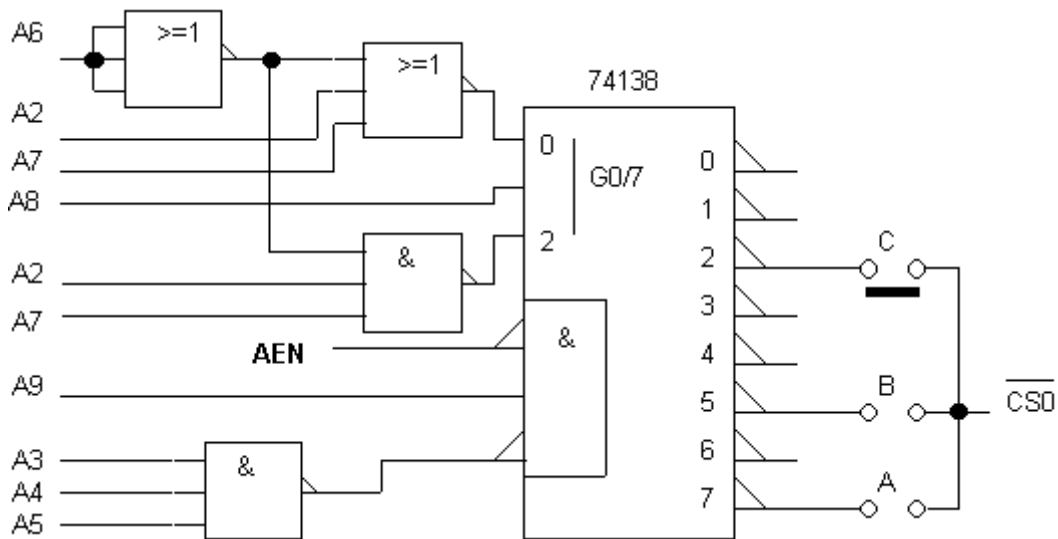
Schéma 1 :

Compléter chaque sortie en indiquant la zone adressée et la capacité correspondante.



Donner l'équation logique de la sortie CS4\.

Schéma 2 :



Indiquer le niveau de AEN (Adress ENable) validant le décodage.

Pour chaque position de cavalier implanté (A OU B OU C) :

- déterminer les plages d'adresses possibles pour lesquelles la sortie est au niveau bas en complétant le tableau ci-dessous.
- indiquer la capacité mémoire correspondante.

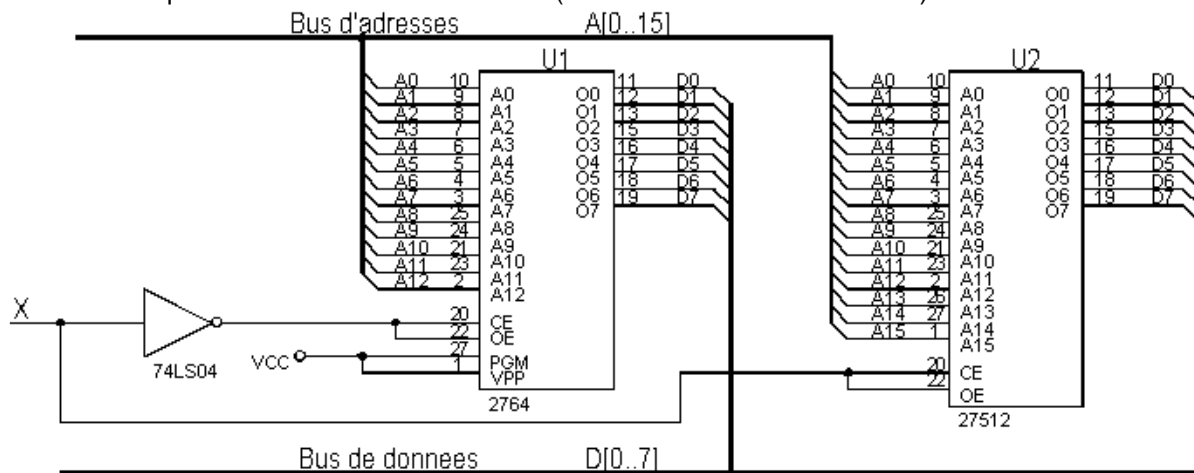
Table de décodage, schéma 2 :

Exercice 3 :

Déterminer pour le schéma ci dessous :

- La capacité mémoire des circuits U1 et U2

L'état de chaque circuit suivant le niveau de X (CE et OE actifs au niveau bas)



6. Les Microcontrôleurs :

Ce sont des circuits intégrés regroupant les fonctions d'un « système » minimum à microprocesseur.

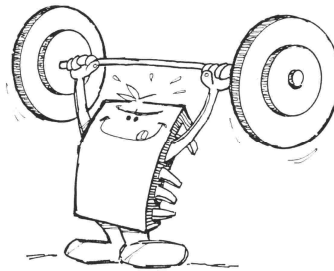
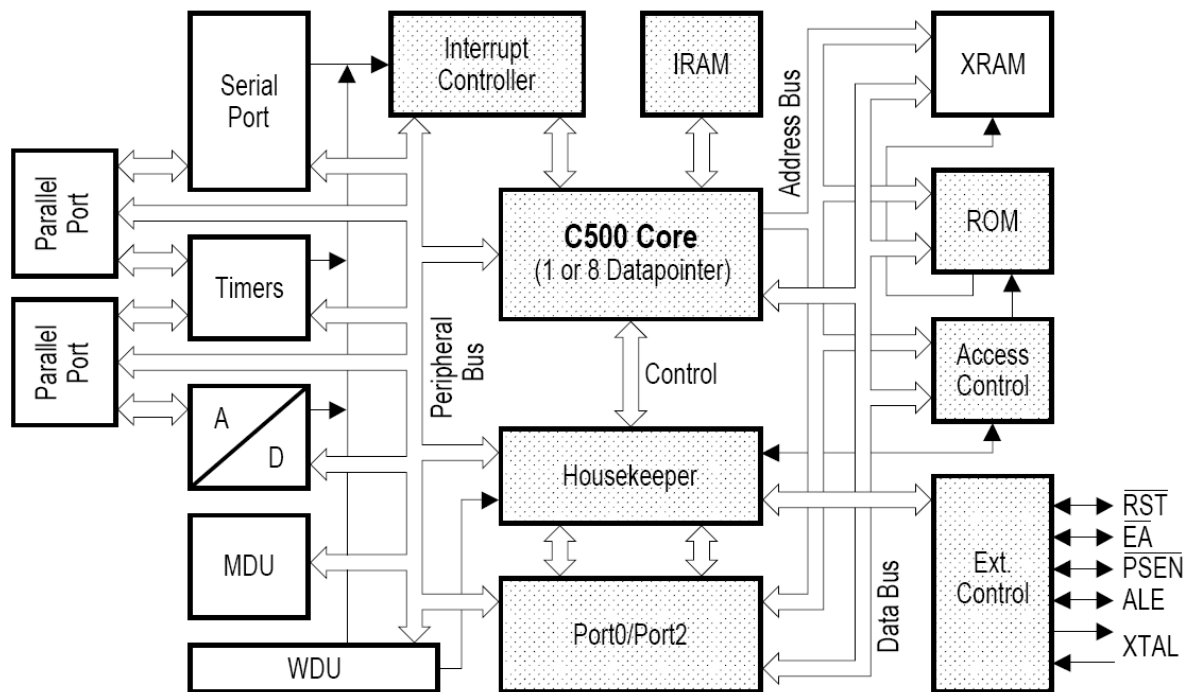
Avantages des microcontrôleurs :

- Intégration dans un seul boîtier
- Fiabilité
- Coût de câblage réduit
- Faible consommation

Inconvénients

L'intégration de nombreux périphériques, de RAM, de ROM limite la puissance de calcul et la vitesse de ces circuits (les transistors intégrés sont destinés aux périphériques et non plus au calcul).
Mise en œuvre et approche du composant d'apparence complexe.

6.1. Organisation générale (INFINEON-SIEMENS C517A, CPU type INTEL 8051)



6.2. L'unité centrale

On retrouve en commun dans toutes les unités centrales (CPU ou CORE):

- Le compteur de programme qui contient l'adresse de la prochaine instruction
- Un ou des registres d'index qui permettent l'adressage indirect (ici Rn ou DPTR)
- Une ou des registres de pile (au moins la pile S) qui pointe des données « empilées » dans le RAM (voir page 40)
- Un registre de code condition donnant entre autre le type de résultat de l'instruction précédente afin de conditionner les instructions de saut.
- Un ou deux accumulateurs (A et B)
- Une unité arithmétique et logique, elle effectue des opérations entre l'accumulateur et l'opérande qui peut être le code suivant l'instruction ou une donnée en mémoire. Le résultat est toujours placé dans l'accumulateur A.
- Un bus de données : 8 ou 16 bits pour les petits « systèmes » permettant les échanges entre les registres, les mémoires et les périphériques
- Un bus d'adresses : en générale 16 bits sur les microcontrôleurs. soit 65536 adresses (64KO)
- Un bus de contrôle permettant de piloter les périphériques et mémoires et d'indiquer l'état du microprocesseur.

6.3. Ports parallèles

Ces ports permettent le transfert d'informations en parallèle, généralement sur 4 ou 8 bits. Ils peuvent être :

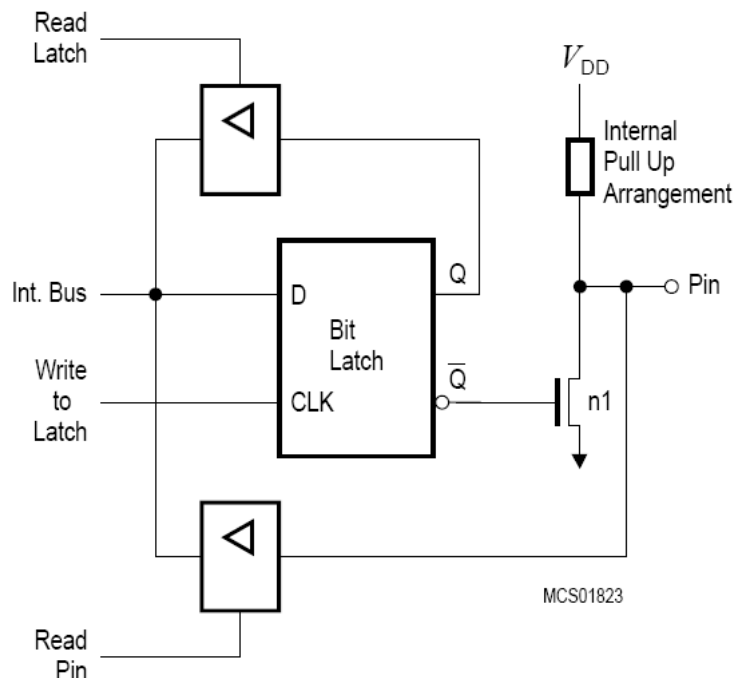
- Toujours en entrée
- Toujours en sortie
- Configurable en Entrée OU en Sortie (dans ce dernier cas ils sont en entrée par défaut).

Leur utilisation est très simple s'ils sont unidirectionnels, il suffit de lire ou d'écrire dans leur registre. Sinon les ports d'E/S doivent être configurés avant leur utilisation.

Si leur sortie est de type push-pull (soit à l'état 1, soit à l'état 0) il existe un registre de direction (Data

Direction Register) permettant de définir le sens de transfert des données.

Si la sortie est à drain-ouvert, le registre DDR est inutile, pour utiliser le port en entrée, il suffit de disposer une résistance pull-up et de placer un '1' en sortie.



Cas du 8051-C517A :

En écrivant un '1' sur le PORT la sortie /Q de la bascule D passe à '0', ce qui bloque n1. La sortie (Pin) peut être utilisée en entrée (il y a seulement une charge pull-up).

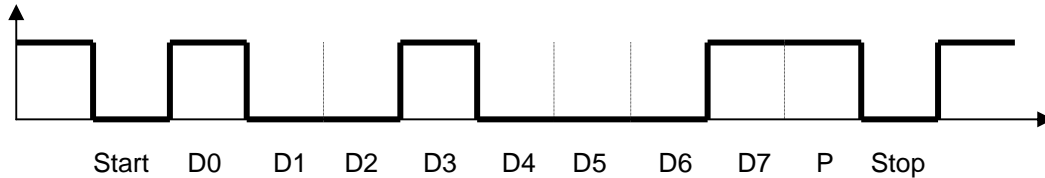
Ex : `mov P1, #0Fh`
permet d'entrer des données sur les bits 3,2,1,0

Rq : par défaut les PORTS de C517A sont à '1' après le RESET.

6.4. Ports séries asynchrones

Ces périphériques⁶ permettent d'établir des communications séries asynchrones suivant le format NZR (No Return to Zero). Ils communiquent avec le microprocesseur en parallèle par l'intermédiaire du bus de données et en série avec l'extérieur par une liaison série asynchrone (sans horloge). Ils peuvent par ailleurs piloter un modem.

Exemple de frame : asynchrone 1start, 8 bits, parité paire, 1 stop : nombre 10010001



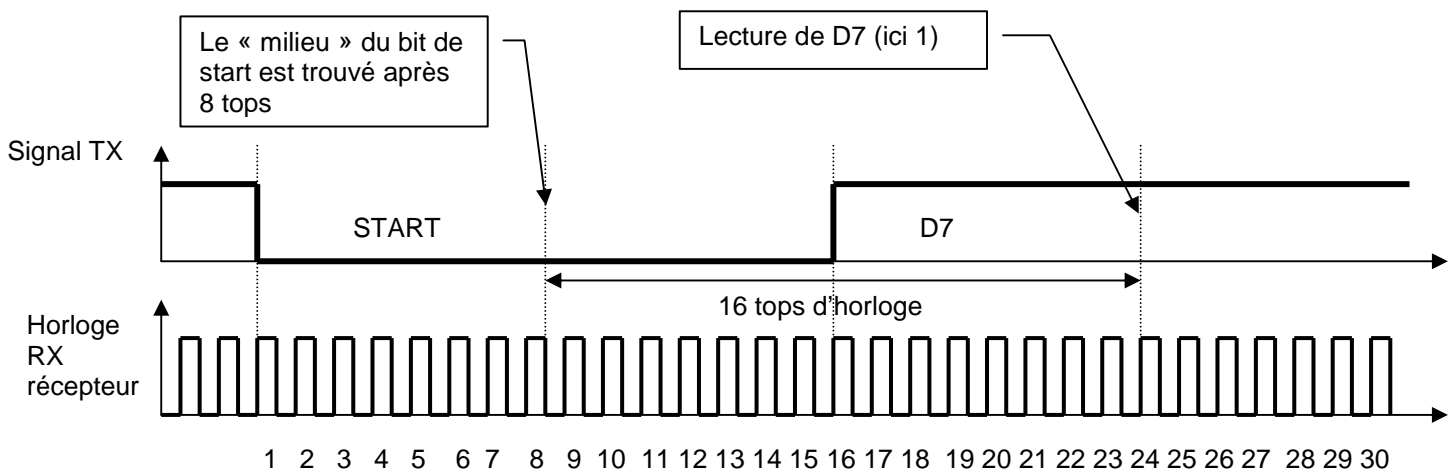
Parité paire : le bit de parité est positionné pour que l'ensemble des bits de donnée et le bit de parité représente un nombre de bits à 1 pair

Parité impaire : le bit de parité est positionné pour que l'ensemble des bits de donnée et le bit de parité représente un nombre de bits à 1 impair

Dans ce type de transmission l'horloge de transmission est comprise dans le signal, le bit de start est utilisé par le récepteur pour se synchroniser avec l'émetteur. Cependant les deux horloges de transmission et de réception sont au départ très proche.

L'horloge de réception possède une fréquence multiple de celle de transmission (en général x16 ou x64). Dans le cas d'une division par 16 :

Lors de la réception du front descendant du bit de start, l'USART attend 8 tops d'horloge, le circuit reçoit alors théoriquement le milieu du bit de start, l' USART attend ensuite 16 tops d'horloge, le circuit reçoit alors le milieu de D7 et lit ce bit, l' USART attend ensuite 16 tops etc. L'horloge du récepteur est donc resynchronisée lors de la réception de chaque caractère.



L'horloge RX (réception) doit donc toujours être supérieure à celle de TX (transmission). En réalité les deux horloges sont identiques et TX est divisé dans l'ACIA pour produire la vitesse de transmission souhaité.

Registres internes d'un USART, Choix du format de transmission

Nombre de bits de données : 7 ou 8

Parité : paire, impaire, sans

Nombre de bits de stop : 1 ou 2

Généralement on rend égale le nombre de bits pour un octet égal à 10. De cette manière il suffit de diviser la vitesse en BAUDS (bits par seconde) par 10 pour obtenir la vitesse en octets par seconde.

Choix de la division d'horloge (1, 16, 64 en général)

⁶ USART : Universal Synchronous Asynchronous Receiver Transceiver
UART : Universal Asynchronous Receiver Transceiver (pour les interfaces uniquement asynchrone)

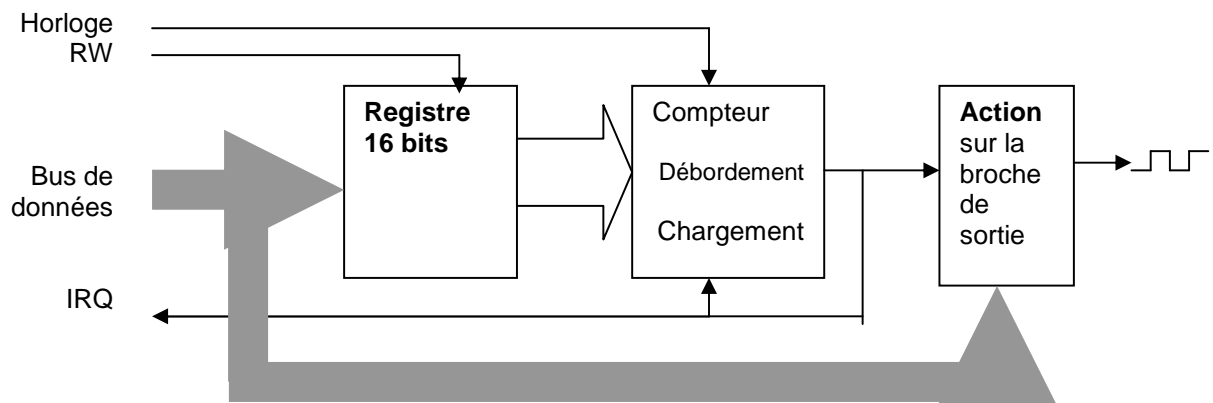
Contrôle des commandes de communication avec handshaking, siganux RTS et CTS (Ready To Send et Clear To Send)

6.5. TIMERS

Ces périphériques permettent de produire ou de mesurer des durées et des fréquences.

6.5.1. Production de signaux, principe :

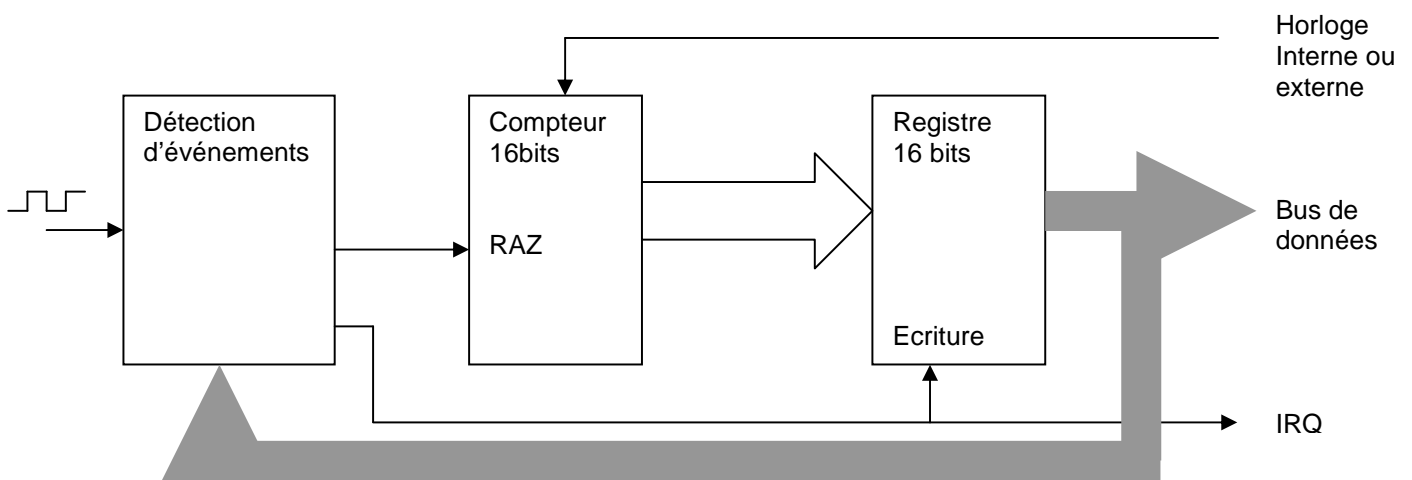
Un compteur 16bits compte depuis une valeur chargée dans un registre et agit sur une broche du circuit lors du passage de 0xFFFF à 0. Le TIMER peut être configuré afin de recharger le compteur lors du débordement. L'horloge pouvant être celle du uP ou une horloge externe.



6.5.2. Mesure de durée, principe :

Un compteur est mis à zéro lors d'un premier événement (front montant ou descendant), lors du deuxième événement la valeur du compteur (le nombre de tops d'horloge comptés) est stockée dans un registre à destination du bus de données.

La durée mesurée est égale à $N \cdot T$. N est le nombre de tops d'horloge compté entre les deux événements et T la période de l'horloge



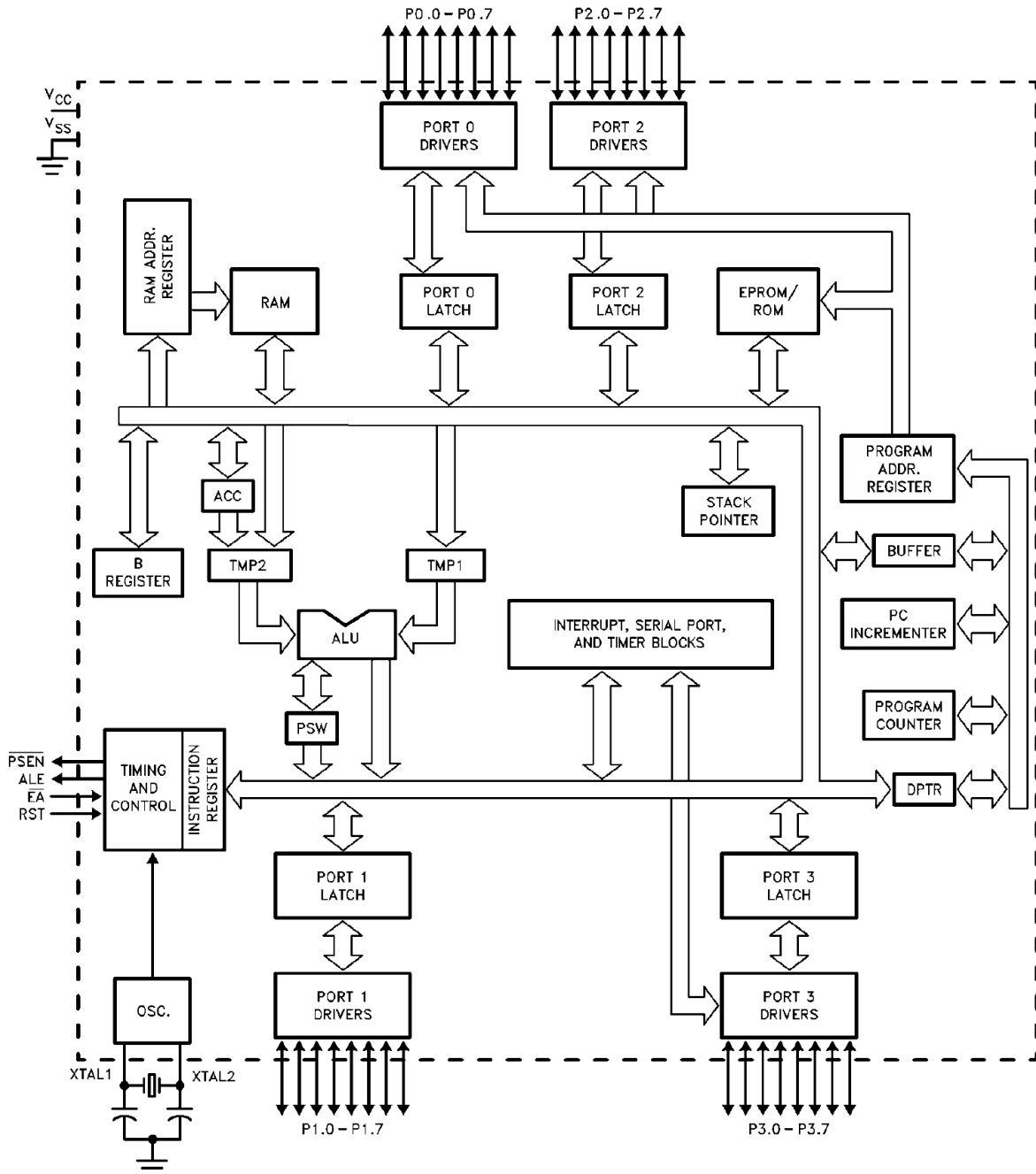
Les timers possèdent de nombreux registres :

Registres 16 bits pour la production ou la mesure de durées

Registres de contrôle pour les actions en sortie. Aucune, mettre à 1, mettre à 0, basculer

Registres de contrôle pour la détection d'événement : Aucune, front montant, descendant, tous Etc.

7. Le standard INTEL 8051.



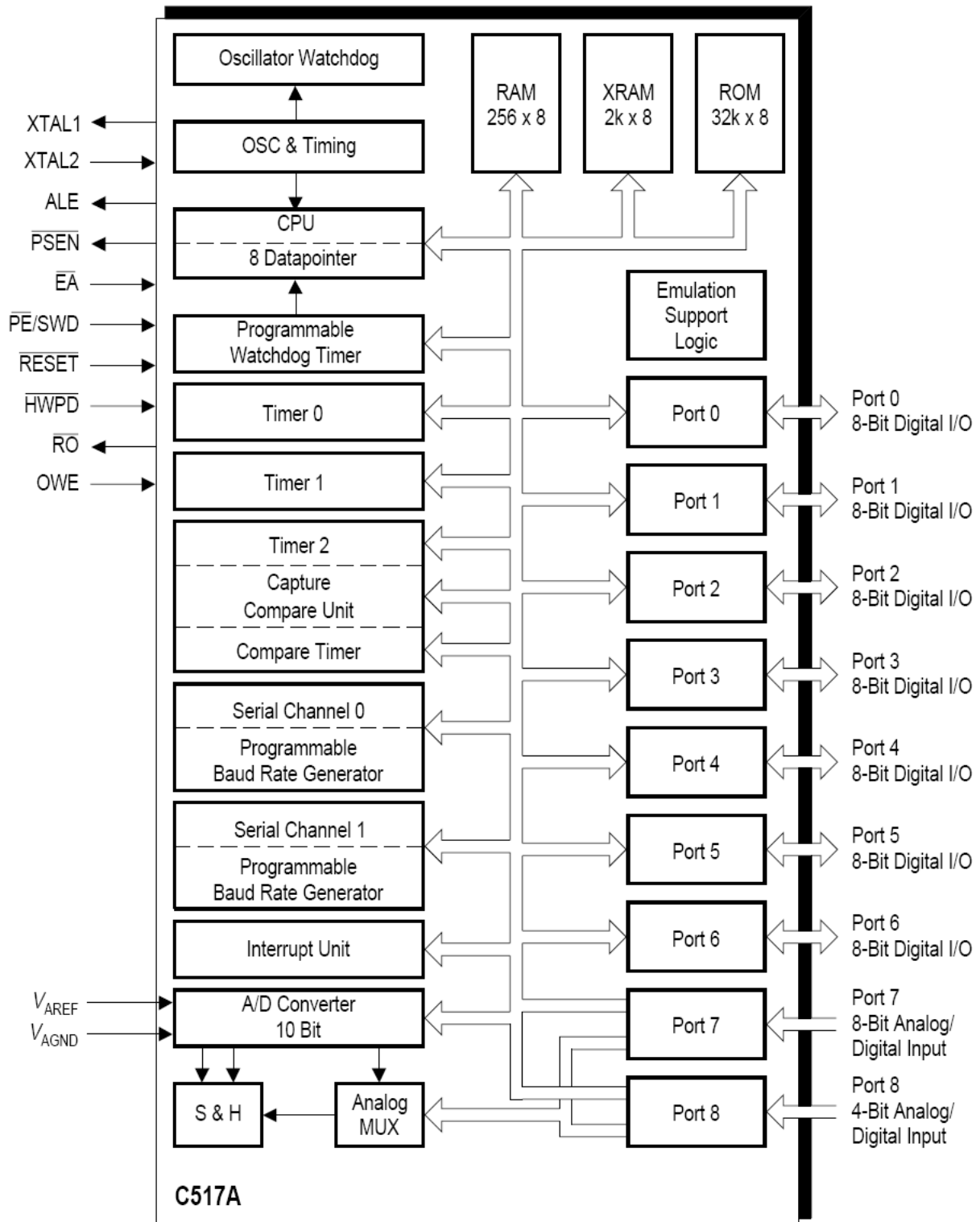
Les microcontrôleurs sont toujours apparemment complexes, cela est du au grand nombre de fonctions qu'ils intègrent (CPU, mémoires, périphériques ...)

On voit ci-dessus communément à tous les microcontrôleurs:

- L'ALU : Arithmetic and Logic Unit
- Les registres accumulateurs (ACC et B)
- Le registre d'état du microcontrôleur (PSW)
- Le pointeur de pile (stack pointer)
- Le gestionnaire de temps et l'oscillateur qui cadence le déroulement du programme
- Le gestionnaire d'interruptions
- Les registres pointeurs (index) DPTR
- Le compteur de programme qui contient l'adresse de l'instruction à exécuter
- Les mémoires RAM, ROM, EPROM
- Les ports parallèles, le port série, les TIMER
- La broche RST (Remise à zéro, active à l'état haut elle initialise le programme)
- ALE, Adresse Latch Enable, utilisée en mode « mémoire externe »
- /EA, si 1 la mémoire interne 0x000 à 0xFFFF est sélectionnée
- /PSEN, indique un accès à la mémoire externe

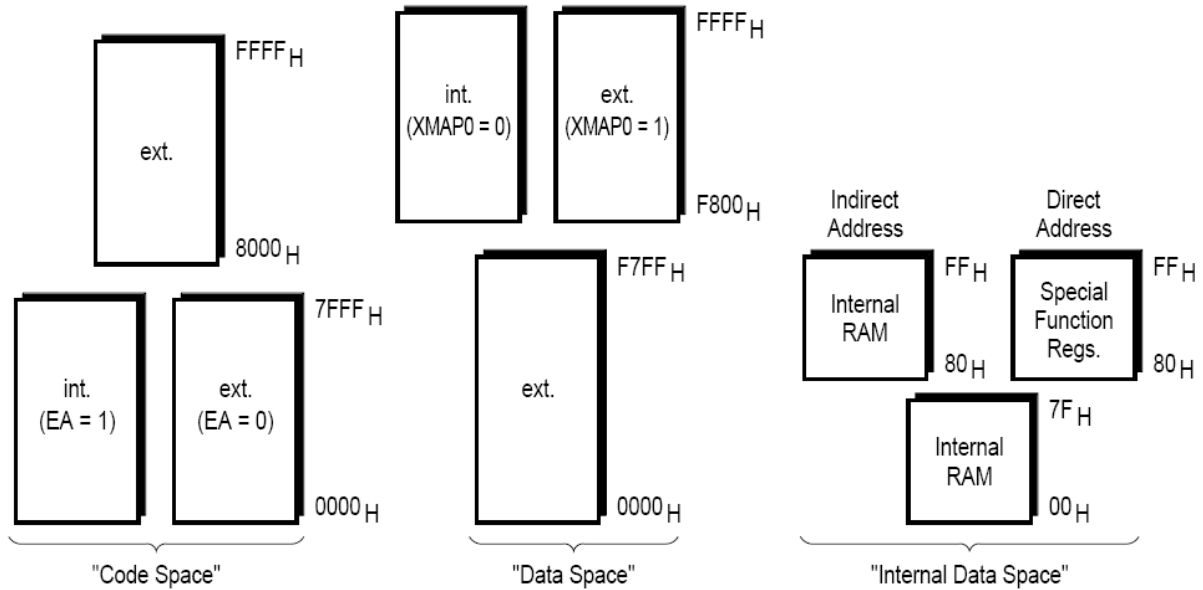
7.1. Un évolution du 8051, l' INFINEON (SIEMENS) C517A

Ce microcontrôleurs possède un noyau (CORE) compatible avec le standard INTEL 8051, il est équipé en plus du 8051 d'un convertisseur analogique numérique, de fonctions « COMPARE », d'un chien de garde



- 8 PORTS parallèles
- Un convertisseur analogique numérique 10 bits
- Deux ports séries asynchrones (USART)
- Trois TIMER
- Un chien de garde (WD)
- 32KO de mémoire programme (ROM)
- 2KO + 256Octets de RAM

7.2. Organisation de la mémoire



Rq : l'utilisation de mémoires externes (ROM ou RAM) limite le nombre de périphériques (qui deviennent des BUS adresses-données)

Mémoire programme (CODE space)

0x0000 à 0x7FFF (32KO) en interne ou 64KO en externe.

Lors d'un RESET le program counter (PC) pointe l'adresse 0x0000 (toujours le début du programme)

Mémoire de données (DATA space) :

- Une RAM externe est adressable de 0x0000 à 0xFFFF
- La mémoire interne est composée de trois parties

Les adresses RAM 0x00 à 0x7F sont adressables en direct et en indirect

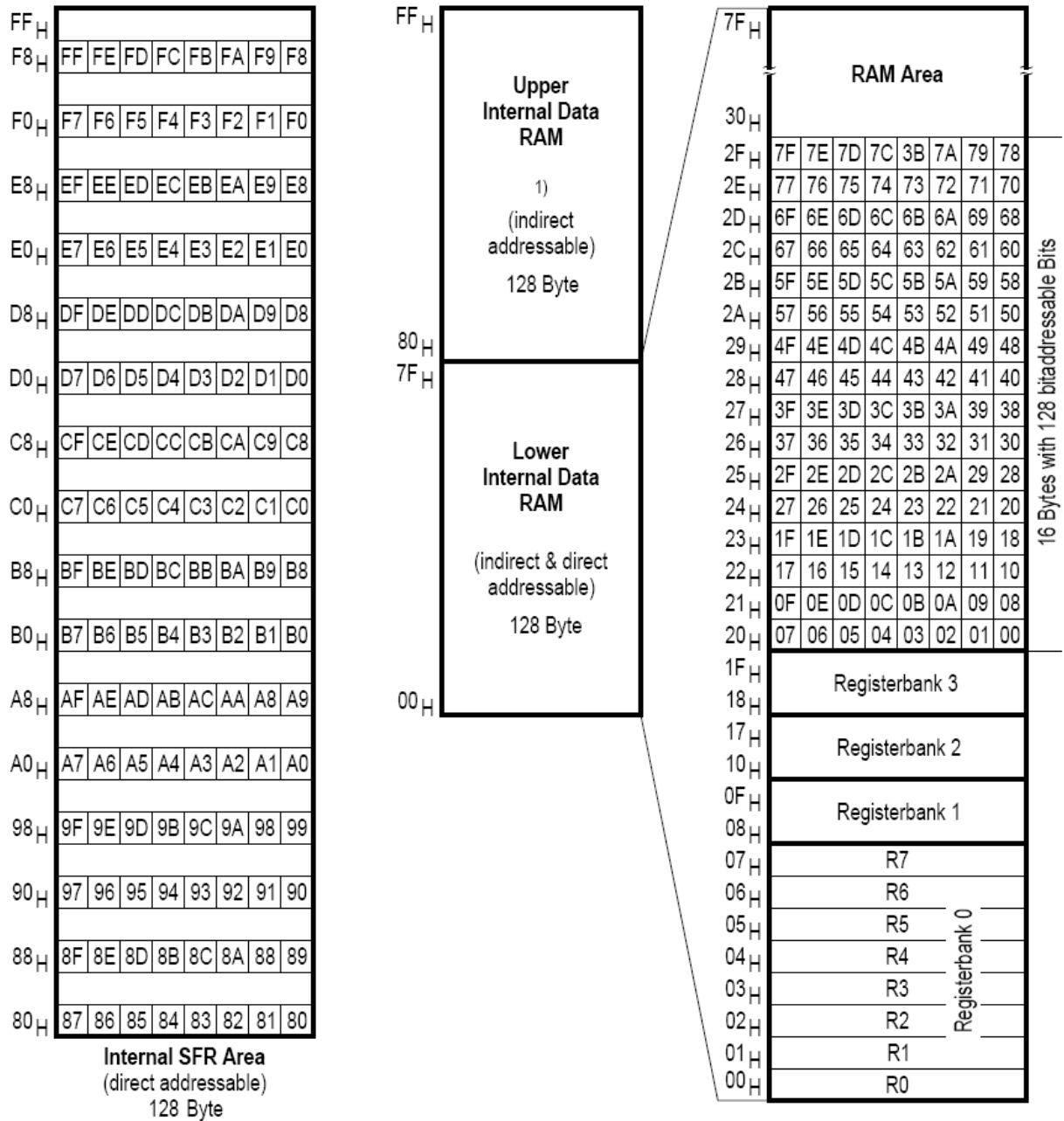
0x00 à 0x1F : 4 banques de 8 registres généraux. (Une seule banque active) les registres R0 et R1 peuvent servir de pointeurs en RAM (adressage indirect).

0x20 à 0x2F : 16 octets soit 128 bits adressables individuellement

0x30 à 0x7F : 80 octets

0x80 à 0xFF : 128 octets adressables uniquement en indirect

0x80 à 0xFF : 128 registres internes et de périphériques adressables uniquement en direct (les adresses finissant par 0 ou 8 sont également adressables par bit)



8051 : Plan mémoire

8051 Special Function Registers

SYMBOL	DESCRIPTION	DIRECT ADDRESS	BIT ADDRESS, SYMBOL, OR ALTERNATIVE PORT FUNCTION								RESET VALUE
			MSB				LSB				
ACC*	Accumulator	E0H	E7	E6	E5	E4	E3	E2	E1	E0	00H
AUXR#	Auxiliary	8EH	-	-	-	-	-	-	-	AO	xxxxxxx0B
AUXR1#	Auxiliary 1	A2H	-	-	-	LPEP ²	WUPD ³	0	-	DPS	xxx000x0B
B*	B register	F0H	F7	F6	F5	F4	F3	F2	F1	F0	00H
DPTR:	Data Pointer (2 bytes)										
DPH	Data Pointer High	83H									00H
DPL	Data Pointer Low	82H									00H
			AF	AE	AD	AC	AB	AA	A9	A8	
IE*	Interrupt Enable	A8H	EA	-	ET2	ES	ET1	EX1	ET0	EX0	0x000000B
			BF	BE	BD	BC	BB	BA	B9	B8	
IP*	Interrupt Priority	B8H	-	-	PT2	PS	PT1	PX1	PT0	PX0	xx000000B
			B7	B6	B5	B4	B3	B2	B1	B0	
IPH#	Interrupt Priority High	B7H	-	-	PT2H	PSH	PT1H	PX1H	PT0H	PX0H	xx000000B
			87	86	85	84	83	82	81	80	
P0*	Port 0	80H	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	FFH
			97	96	95	94	93	92	91	90	
P1*	Port 1	90H	-	-	-	-	-	-	T2EX	T2	FFH
			A7	A6	A5	A4	A3	A2	A1	A0	
P2*	Port 2	A0H	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	FFH
			B7	B6	B5	B4	B3	B2	B1	B0	
P3*	Port 3	B0H	RD	WR	T1	T0	INT1	INT0	TxD	RxD	FFH
PCON# ¹	Power Control	87H	SMOD1	SMOD0	-	POF	GF1	GF0	PD	IDL	00xx0000B
			D7	D6	D5	D4	D3	D2	D1	D0	
PSW*	Program Status Word	D0H	CY	AC	F0	RS1	RS0	OV	-	P	000000x0B
RACAP2H#	Timer 2 Capture High	CBH									00H
RACAP2L#	Timer 2 Capture Low	CAH									00H
SADDR#	Slave Address	A9H									00H
SADEN#	Slave Address Mask	B9H									00H
SBUF	Serial Data Buffer	99H									xxxxxxx0B
			9F	9E	9D	9C	9B	9A	99	98	
SCON*	Serial Control	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	T1	R1	00H
SP	Stack Pointer	81H									07H
			8F	8E	8D	8C	8B	8A	89	88	
TCON*	Timer Control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	00H
			CF	CE	CD	CC	CB	CA	C9	C8	
T2CON*	Timer 2 Control	C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2	00H
T2MOD#	Timer 2 Mode Control	C9H	-	-	-	-	-	-	T2OE	DCEN	xxxxxx00B
TH0	Timer High 0	8CH									00H
TH1	Timer High 1	8DH									00H
TH2#	Timer High 2	CDH									00H
TL0	Timer Low 0	8AH									00H
TL1	Timer Low 1	8BH									00H
TL2#	Timer Low 2	CCH									00H
TMOD	Timer Mode	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	00H

* SFRs are bit addressable.

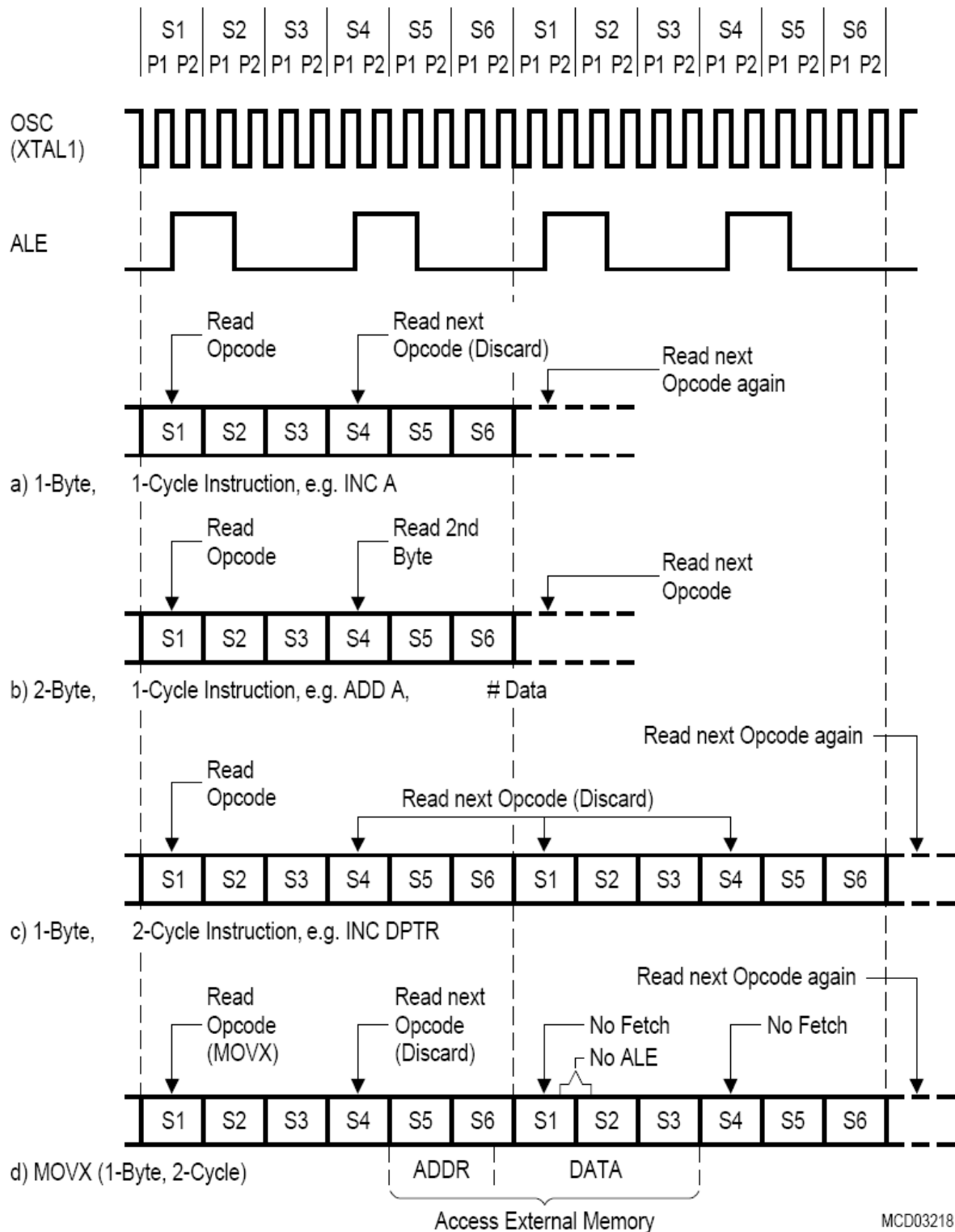
SFRs are modified from or added to the 80C51 SFRs.

- Reserved bits.

1. Reset value depends on reset source.

2. LPEP – Low Power EPROM operation (OTP/EPROM only)

8051 : recherche, décodage, exécution d'une instruction



Chaque partie du cycle « recherche, décode, opérande exécute » coûte deux impulsions d'horloge, il faut 12 ou 24 impulsions d'horloge pour exécuter une instruction. Ainsi un 8051 cadencé à 24MHz aura une horloge d'instruction de 2MHz (500nS par instruction)

7.2.1. Unité centrale 8051, éléments d'assembleur

ACC est l'accumulateur. Il comprend 8 bits.

```
mov a,# 12 ; charge la valeur 12 dans le registre ACC
```

B est un accumulateur utilisé pour les opérations de multiplication et de division (nécessitant 16bits)

```
mul AB AxB , ;résultats pFORTS dans B, pfaibles dans ACC
div AB A/B, ;résultat dans ACC, reste dans B
```

Les registres R0 et R1 servent comme registres d'index (pointeurs) pour adresser la mémoire RAM

```
mov a,70h ;Charge 70h dans ACC
mov 71h,a ;enregistre ACC dans la case 71h
```

Au lieu de cette séquence on peut écrire

```
mov R0,70h ;R0 pointe l'adresse 70h
mov a,@R0 ;le contenu de 70h est placé dans ACC
inc R0 ;R0 pointe l'adresse 71h
mov @R0,a ;le contenu de ACC est placé à l'adresse 71h
```

Le pointeur de pile SP contient l'adresse du premier octet, qui n'est pas encore réservé par la pile.

Le registre d'instructions PC contient l'adresse de l'instruction à exécuter.

Registre PSW : Program Status Word :

Le registre PSW contient huit bits. Beaucoup d'opérations changent ces bits. Quelques opérations changent leur comportement selon ces bits. Par exemple

```
inc a ajoute 1 à ACC, change dans PSW les bits CY, AC,OV,P
```

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	F1	P

CY Carry, Retenue: Le résultat de la dernière opération a causé une retenue. Le bit permet des opérations sur des opérandes plus longues que 8bits.

AC auxiliary carry : retenue pour les chiffres de 4 bits (codage BCD)

OV Overflow, Débordement: Le résultat de la dernière opération a causé un débordement. Seulement pour des nombres entiers signés complément à 2.

F0 : drapeau (flag) d'usage général

RS1, RS0 : sélection de la banque R0-R7 active

F1 : drapeau (flag) d'usage général

P, parity : si égale à '1', indique que le nombre de bits à '1' de ACC est pair.

7.2.2. Modes d'adressage

Exemples

```
ADD A,7FH (adressage direct) A=A+contenu adresse 7F
ADD A,@R0 (adressage indirect) A=A+contenu de l'adresse pointé par R0
ADD A,R7 (adressage registre) A=A+contenu de R7
ADD A,# 127 (adressage immédiat) A=A+127
```

Registre : toute opération entre un registre (R0-R7) et ACC, B, DPTR.

```
mov a,R0 ;copie la valeur dans R0 dans ACC
mov R1,a ;copie la valeur dans ACC dans R1
mov R1,R0 ;copie la valeur dans R0 dans R1
```

Immédiat: L'opérande se trouve directement dans le programme derrière le code de l'instruction. Au niveau d'assemblage, ce mode d'adressage s'exprime avec le symbole dièse (#).

```
mov a,#12 ;place 12 dans ACC
```

Direct : l'adresse de l'opérande se trouve dans l'octet suivant le code de l'instruction. Ce mode d'adressage permet d'accéder aux registres SFR (périphériques, etc.) ainsi qu'au 128 octets hauts de la RAM interne.

```
mov 85h,a ;copie ACC à l'adresse 85h
```

Registre indirect : ou adressage indexé, réservé aux accès en RAM. L'adresse de la donnée est contenue dans les registres R0, R1, SP pour l'accès RAM et R0, R1, DPTR pour l'accès ROM. L'indirection est marquée par un '@'. L'adresse est sur 8 bits (0x00 à 0xFF)

```
mov R0,85h ; R0 pointe 85h
mov @R0,a ; copie ACC à l'adresse 85h
```

Base register plus index register addressing : permet d'accéder aux données en ROM. L'adresse de la donnée est calculée en ajoutant le décalage ACC au registre DPTR (data pointer) ou PC. Ceci permet d'accéder à toutes les adresses

```
clr a ; ACC=0
mov DPTR,#0A34h ; DPTR=A34h
movc a,@a+DPTR ; ACC egale le contenu de la mémoire ROM A34h
```

7.2.3. Le processeur Booleen :

Le jeu d'instruction du 8051 permet d'accéder directement à certains bits (SFR, RAM, PSW).

- mettre un bit à 1
- mettre un bit à 0
- complémenter un bit
- saut si le bit est à 1
- saut si le bit est à 0
- saut si le bit est à 1 puis passer le bit à 0
- déplacer le bit



8. Le jeu d'instructions du 8051 :

Syntaxe:

Rn – registres R0-R7

@Ri – adressage indirect RAM interne ou externe par les registres R0 ou R1

#data 8 – adressage immédiat (8 ou 16 bits)

bit – 128 drapeaux généraux et bits des SFR

A – Accumulateur ACC

Mnemonic	Description	Byte	Cycle
Logic Operations			
ANL	A,Rn	AND register to accumulator	1 1
ANL	A,direct	AND direct byte to accumulator	2 1
ANL	A,@Ri	AND indirect RAM to accumulator	1 1
ANL	A,#data	AND immediate data to accumulator	2 1
ANL	direct,A	AND accumulator to direct byte	2 1
ANL	direct,#data	AND immediate data to direct byte	3 2
ORL	A,Rn	OR register to accumulator	1 1
ORL	A,direct	OR direct byte to accumulator	2 1
ORL	A,@Ri	OR indirect RAM to accumulator	1 1
ORL	A,#data	OR immediate data to accumulator	2 1
ORL	direct,A	OR accumulator to direct byte	2 1
ORL	direct,#data	OR immediate data to direct byte	3 2
XRL	A,Rn	Exclusive OR register to accumulator	1 1
XRL	A direct	Exclusive OR direct byte to accumulator	2 1
XRL	A,@Ri	Exclusive OR indirect RAM to accumulator	1 1
XRL	A,#data	Exclusive OR immediate data to accumulator	2 1
XRL	direct,A	Exclusive OR accumulator to direct byte	2 1
XRL	direct,#data	Exclusive OR immediate data to direct byte	3 2
CLR	A	Clear accumulator	1 1
CPL	A	Complement accumulator	1 1
RL	A	Rotate accumulator left	1 1
RLC	A	Rotate accumulator left through carry	1 1
RR	A	Rotate accumulator right	1 1
RRC	A	Rotate accumulator right through carry	1 1
SWAP	A	Swap nibbles within the accumulator	1 1

Data Transfer				
MOV	A,Rn	Move register to accumulator	1	1
MOV	A,direct	Move direct byte to accumulator	2	1
MOV	A,@Ri	Move indirect RAM to accumulator	1	1
MOV	A,#data	Move immediate data to accumulator	2	1
MOV	Rn,A	Move accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct byte	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load data pointer with a 16-bit constant	3	2
MOVC	A,@A + DPTR	Move code byte relative to DPTR to accumulator	1	2
MOVC	A,@A + PC	Move code byte relative to PC to accumulator	1	2
MOVX	A,@Ri	Move external RAM (8-bit addr.) to A	1	2
MOVX	A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVX	@Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVX	@DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with accumulator	1	1
XCH	A,direct	Exchange direct byte with accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with accumulator	1	1
XCHD	A,@Ri	Exchange low-order nibble indir. RAM with A	1	1

Boolean Variable Manipulation

CLR	C	Clear carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set carry flag	1	1
SETB	bit	Set direct bit	2	1
CPL	C	Complement carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to carry flag	2	2
ANL	C,/bit	AND complement of direct bit to carry	2	2
ORL	C,bit	OR direct bit to carry flag	2	2
ORL	C,/bit	OR complement of direct bit to carry	2	2
MOV	C,bit	Move direct bit to carry flag	2	1
MOV	bit,C	Move carry flag to direct bit	2	2

Program and Machine Control

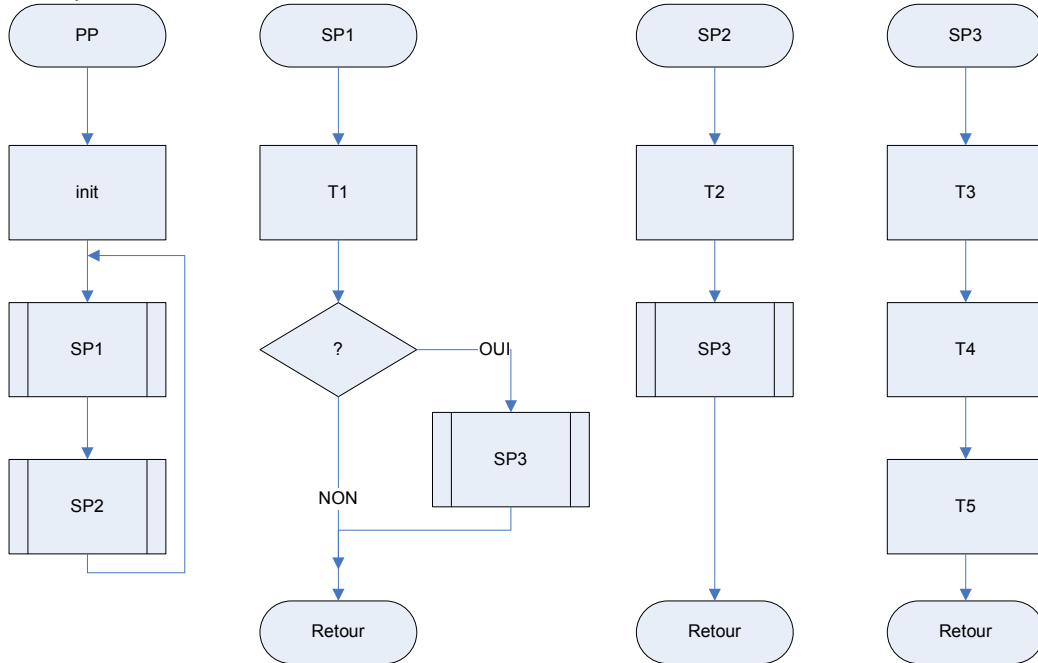
ACALL	addr11	Absolute subroutine call	2	2
LCALL	addr16	Long subroutine call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute jump	2	2
LJMP	addr16	Long iump	3	2
SJMP	rel	Short jump (relative addr.)	2	2
JMP	@A + DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if accumulator is zero	2	2
JNZ	rel	Jump if accumulator is not zero	2	2
JC	rel	Jump if carry flag is set	2	2
JNC	rel	Jump if carry flag is not set	2	2
JB	bit,rel	Jump if direct bit is set	3	2
JNB	bit,rel	Jump if direct bit is not set	3	2
JBC	bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE	A,direct,rel	Compare direct byte to A and jump if not equal	3	2
CJNE	A,#data,rel	Compare immediate to A and jump if not equal	3	2
CJNE	Rn,#data rel	Compare immed. to reg. and jump if not equal	3	2
CJNE	@Ri,#data,rel	Compare immed. to ind. and jump if not equal	3	2
DJNZ	Rn,rel	Decrement register and jump if not zero	2	2
DJNZ	direct,rel	Decrement direct byte and jump if not zero	3	2
NOP		No operation	1	1

9. Programmation structurée et sous-programmes

Lors de la création d'un programme il faut s'assurer de :

- Sa lisibilité
- Sa maintenabilité (découper le programme en fonctions logicielles ou sous-programmes)
- Sa compacité (éviter les groupes d'instructions redondants)

Exemple :



Il est nécessaire de découper le logiciel en un programme principal associé à un ensemble de sous-programmes, afin d'éviter les redondances et de faciliter la lecture.

Les microcontrôleurs disposent d'un registre dédié à la sauvegarde du PC (program counter) lors de l'appel d'un sous programme (Stack Pointer ou SP). Ce registre pointe une adresse en RAM qui est automatiquement incrémenté lors d'une sauvegarde et décrétementée lors de la récupération, c'est une pile LIFO (Last In First Out). Le gros avantage est l'économie de place en RAM, une même adresse peut contenir des données de types différents suivant le contexte du programme. En début de programme le SP doit être initialisé avec une adresse RAM disposant de suffisamment d'espace. L'appel d'un sous programme s'effectue par l'instruction call adresse (sur 12 bits) ou lcall adresse (sur 16 bits)

L'adresse de retour est mémorisée dans la pile S (SP est incrémenté deux fois). Le compteur de programme (PC) est chargé avec l'adresse de destination. Le sous programme se termine par l'instruction RET qui a pour effet de recharger le PC avec l'adresse de retour stockée dans la pile (SP est décrétementé deux fois).

La pile permet également la sauvegarde rapide de données par les instructions push et pop. Attention il doit TOUJOURS y avoir un RET après un CALL et toujours un POP après un PUSH !

call tempo se trouve à l'adresse 0x123 (l'instruction occupe 2 octets)

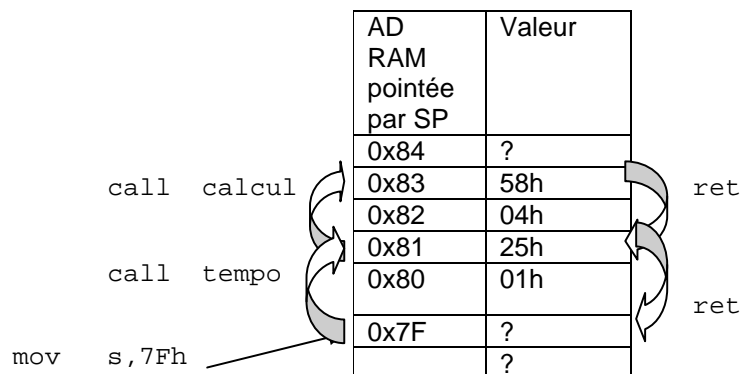
call calcul se trouve à l'adresse 0x456

```

mov    s, 7Fh
mov    a, #55h
call   tempo
mov    a, #12h
...
jmp    ailleurs

tempo:
...
call   calcul
ret

calcul:
...
ret
    
```



10. L'assembleur 8051

Règles de programmation en assembleur :

- Trois « champs » : étiquettes (adresses), instructions, opérandes, commentaires
- Préciser la zone mémoire à utiliser (ROM, RAM, XRAM ...)

Exemple de programme assembleur

```

#include <reg517A.inc>
; CD 6/08
; PROGRAMME FLASH LED PORT20
; ce programme fait clignoter (flash) le port P2.0
;
LED          EQU    P2.0          ; utilisation de la LED sur P2.0
             CSEG  AT 0           ; adresse d'assemblage (ici 0h, adresse de RESET)
             MOV   SP,#7Fh        ; charge pointeur de pile avec la base RAM
SUITE:       CPL   LED            ; complémente la LED (FLASH)
             CALL  TEMPO          ; tempo
             SJMP  SUITE          ; on recommence
; TEMPO LONGUE R0xR1
TEMPO:       MOV   R0,#0FFh       ; charge R0 et R1
TEMPO1:      MOV   R1,#0FFh
TEMPO2:      DJNZ R1,TEMPO2       ; decremente R1 jusqu'à 0
             DJNZ R0,TEMPO1       ; decremente R0 et recharge R1 si R0!=0
             RET
             END
  
```

'#' indique une directive de compilation, ici l'inclusion du fichier reg517A.inc qui contient les définition des registres et périphérique du microcontrôleur utilisé (En particulier l'adresse du bit P2.0)
 ';' indique un commentaire, INDISPENSABLE à la lecture et compréhension du programme

La première colonne contient des étiquettes qui représentent des équivalences ou des adresses (dans ce cas elles sont suivies d'un ':').

La deuxième colonne contient une directive ou une instruction. EQU correspond à une équivalence (ici entre P2.0 s'appellera LED), END indique la fin du programme. MOV, CPL, CALL etc sont des instructions assembleur.

CSEG indique le segment de mémoire courant pour le PROGRAMME (donc la ROM). AT 0 place les codes machines suivants à partir de l'adresse 0 (adresse de RESET).

Rq : l'écriture en colonnes n'est pas obligatoire, un espace suffit pour séparer les champs mais le programme devient alors beaucoup moins visible.

Les directives d'assemblage :

```

Commentaire :           ;
Equivalence :           EQU   var1          EQU   123
Réserve en RAM :       DS    label :        DS    5          ; réserve 5 octets en
RAM
Initialisation de constantes : DB   ici :    B    27,33h,'coucou'
Initialisation de const 16bits : DW   la :    DW    123 ,ABCDh
Réserve d'un bit       DBIT  flag :        DBIT  2          ; reserve 2 bits
Adresse courante       $          jmp   $          ; boucle sans fin
  
```


10.1. Utilisation du Linker – la définition des segments

L'outil de développement permet de s'affranchir de la connaissance des adresses des différentes mémoires du microcontrôleur, cela est utile en raison du grand nombre de cibles disponibles et de leurs configurations mémoires différentes. (Rq : les RAM DATA et IDATA sont communes à tous les microcontrôleurs, mais XDATA et la ROM peuvent être très différents)

La directive « segment » permet de définir un type de segment mémoire. Les types de segments reconnus sont :

DATA : mémoire RAM et SFR de 0x00 à 0xFF, adressable en direct et indirect

BIT : zone RAM « bitadressable » et certains SFR, adressables avec les instructions pour bits.

IDATA : RAM adressable en indirect par les registres R0,R1, (généralement 0x00 à 0xFF)

XDATA : RAM externe accessible par l'instruction MOVX via le registre DPTR.

CODE : ROM accessible via MOVX et le registre DPTR

Nommer un segment : SEGMENT

Sélectionner un segment : RSEG

Segment pré-définis : CSEG (ROM), DSEG (DATA), BSEG(BIT), ISEG (IDATA), XSEG (XDATA)

La directive ORG permet de changer d'adresse DANS un segment

Le linker choisit automatiquement les adresses de différentes zones, le programmeur peut forcer l'adressage (exemple pour le RESET, le programme DOIT commencer à l'adresse 0x0000)

```

1      mesdonnees      SEGMENT DATA      ;mesdonnees est un segment de DATA
2      monprog         SEGMENT CODE          ; ROM
3      mesconst        SEGMENT CODE          ; ROM
4      pile            SEGMENT IDATA         ; creation d'une pile
5
----
6      RSEG      mesdonnees      ;
0000 7      val :          DS      1          ; réserve un octet en RAM
8
----
9      RSEG      pile            ; reservation pour pile
0000 10     DS      20h
11
----
12     RSEG      monprog         ;
0000 13     ORG      0          ; force segment adresse 0
14     ;          CSEG      AT 0          ; autre methode
0000 020000 F 15     LJMP     debut          ; saut sur debut
16
----
17     RSEG      monprog         ; adresse connue du linker !!!
0003 758100 F 18     debut:  MOV     SP,#pile-1
0006 74AB      19     mov     a,#0ABh
0008 F500      F 20     mov     val,a
21
----
22     RSEG      mesconst        ;
0000 1B33636F 23     const:  DB      27,33h,'coucou',0
0004 75636F75
0008 00
24
----
25     RSEG      mesdonnees      ;
0001 26     donnee :  DS      20          ; reserve 20 octets en RAM
27
----
28     RSEG      monprog         ; le programme sera
assemble à la suite du précédent.
000A 04      29     inc     a
000B A800      F 30     mov     R0,val
000D F6      31     mov     @R0,a
000E 80FE      32     jmp     $
33     END

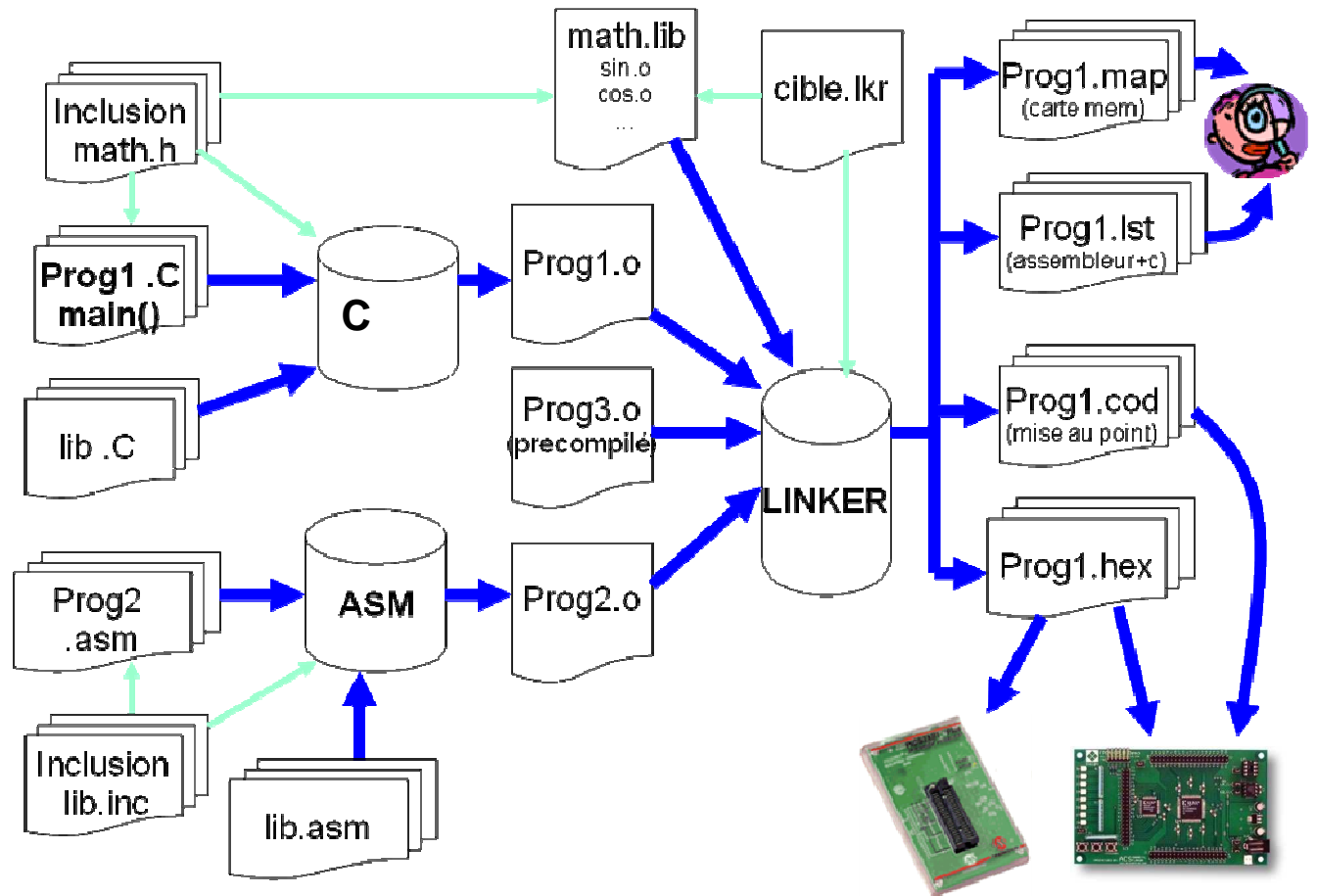
```

11. L'environnement de développement

Généralement l'environnement de développement logiciel comporte :

- Un gestionnaire de projet (choix du compilateur, de la cible, association des fichiers source, configuration écran etc...
- Un assembleur
- Un compilateur C
- Un débogueur pour la mise au point
- Un simulateur
- Un gestionnaire de communications pour le transfert des programmes dans a cible.
- Des outils d'aide au développement (configuration graphique, gestion d'un noyau temps réel...)

Outil de développement : Flux des données



Les outils de développement permettent la création des programmes à partir de fichiers sources écrits en langage C et en assembleur.

Le compilateur C et l'assembleur produisent à partir de plusieurs sources des fichiers objets (*.o) contenant les codes machine de la cible SANS les adresses.

Le fichier *.lkr est propre à chaque cible, il décrit le plan mémoire ainsi que les réservations et éventuellement les fichiers à lier automatiquement.

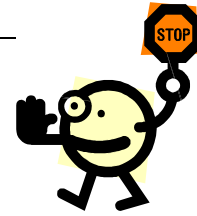
Les fichiers *.lib contiennent un ensemble de fichiers *.o et forment ainsi une bibliothèque (library)

Le linker « lie » les fichiers *.o et *.lib et ajoute les adresses en fonction de son fichier de configuration (*.lkr). Il produit le fichier exécutable (*.hex) et le fichier des symboles pour le debug (*.cod), le fichier listing (*.lst) le plan mémoire (*.map).

Le fichier *.hex est utilisé pour programmer la mémoire de la cible à l'aide d'un programmeur

(versions ROM OTP) ou d'un système interne au microcontrôleur, l'ISP (In Situ Programming)

Certains composants intègrent un gestionnaire de debug, on parle alors d'ICD (In Circuit Debug)



12. Interruptions

12.1. Principes

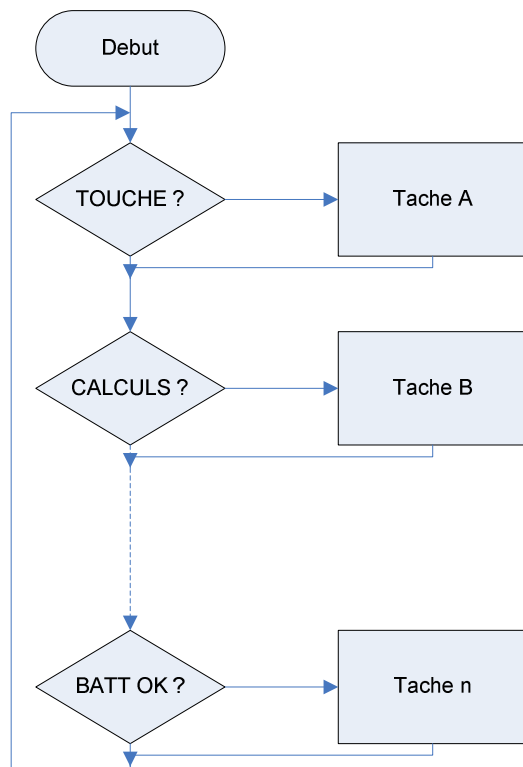
Généralement un programme évolue en fonction d'événements physiques. (touche enfoncée, gestion d'un afficheur, dépassement d'une température, réception d'une communication série...)

Les microcontrôleurs scalaires n'exécutent qu'une instruction à la fois ! Ils ne sont pas réellement multitâches. (C'est un concept qui sera étudié plus tard).

Prenons l'exemple d'une simple calculatrice. Le microcontrôleur doit :

- Détecter l'appuie sur les touches.
- Effectuer les calculs demandés
- Afficher le résultat
- Surveiller l'état des batteries

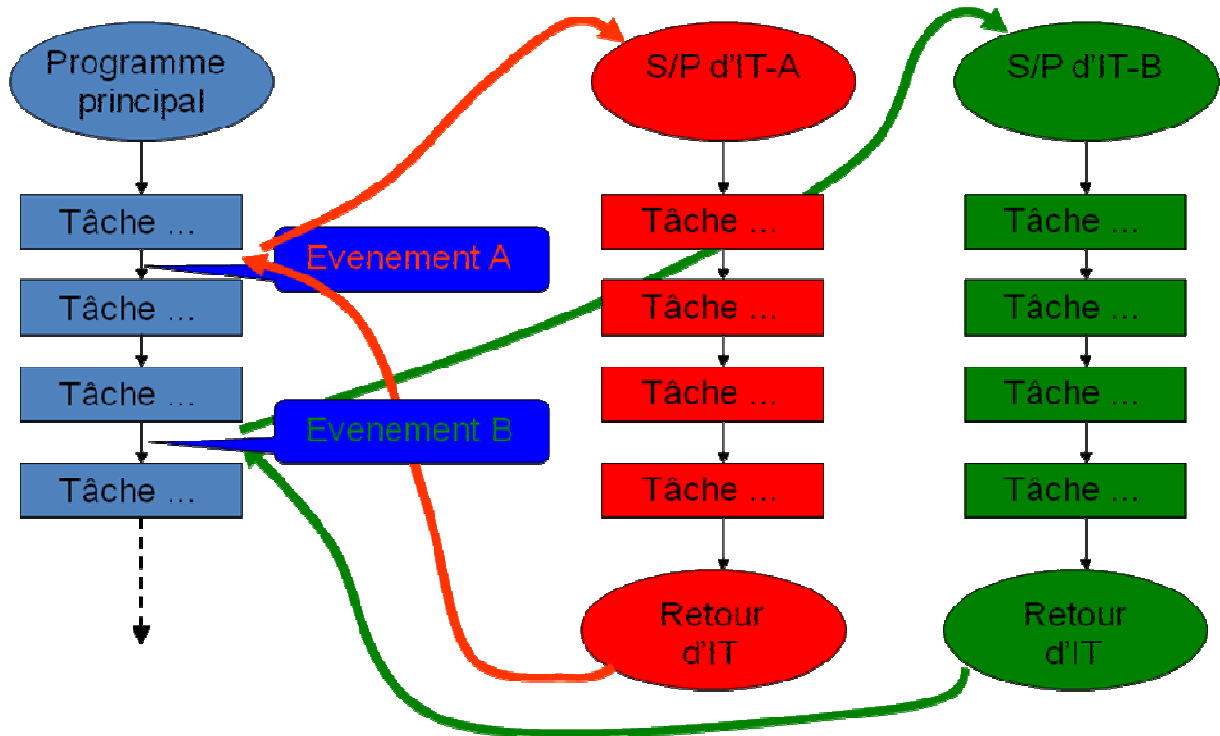
Une première approche consiste à rechercher les tâches à effectuer par scrutation :



On voit que le programme passe beaucoup de temps à surveiller s'il se passe quelque chose, le délai entre l'événement et son traitement n'est pas connu. D'autre part si un événement apparaît durant l'exécution d'une tâche, son traitement en sera d'autant plus retardé.

Une deuxième approche consiste à utiliser le processus d'interruption équipant tous les microcontrôleurs.

Les événements ne sont plus scrutés par programme, mais leur source est connectée à un dispositif **matériel** (interne au microcontrôleur) qui détectera l'événement et interrompra le programme en cours pour effectuer la tâche logicielle correspondante.

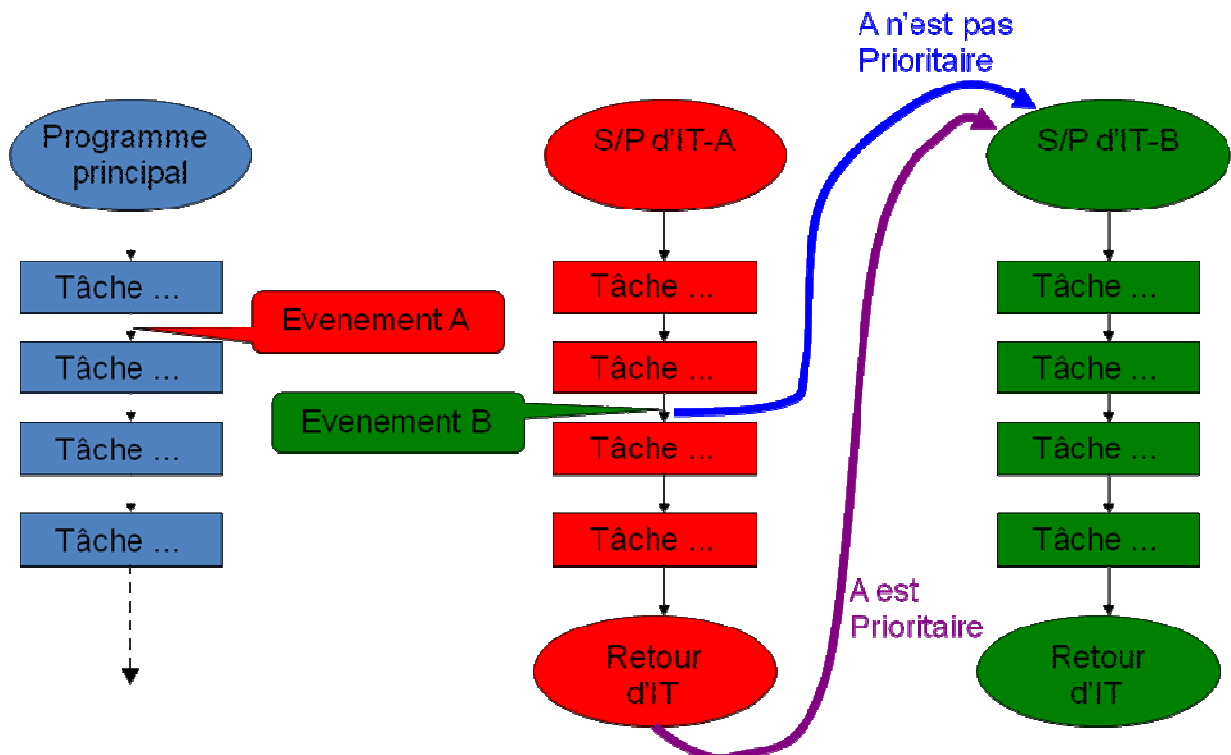


Les avantages de cette technique sont évidents. Le microcontrôleur peut passer son temps à ne rien faire (ou) traiter des tâches permanentes et non prioritaires, il est sollicité automatiquement lorsqu'une nouvelle tâche est nécessaire. Le temps de réponse à un événement est plus rapide et surtout il est connu.

Les contraintes :

- Gestion automatique des adresses des sous-programmes d'interruption (utilisation de vecteurs)
- Le programme en cours peut être interrompu n'importe quand. (un événement physique est asynchrone)
- Les données en cours de traitement avant l'interruption, peuvent être perdues.
- Que se passe t il s'il y a un interruption durant le traitement d'une autre ?

Priorités, soit la tâche en cours est prioritaire sur la nouvelle, soit elle ne l'est pas :



Sauver le contexte :

Lors de l'interruption, le compteur de programme (PC) est chargé avec l'adresse de l'interruption. En fin de le PC doit être chargé avec l'adresse de retour, cette dernière est mémorisée dans la pile S. Si des registres sont modifiés dans le sous-programme d'interruption ils doivent être sauvegardés puis restaurés avant le retour. Ces opération se fond à l'aide d'instructions push ou pull qui stockent des données dans la pile.

ATTENTION, dans une pile LIFO les opérations push et pull doivent être effectuées dans un ordre inverse.

Exemple :

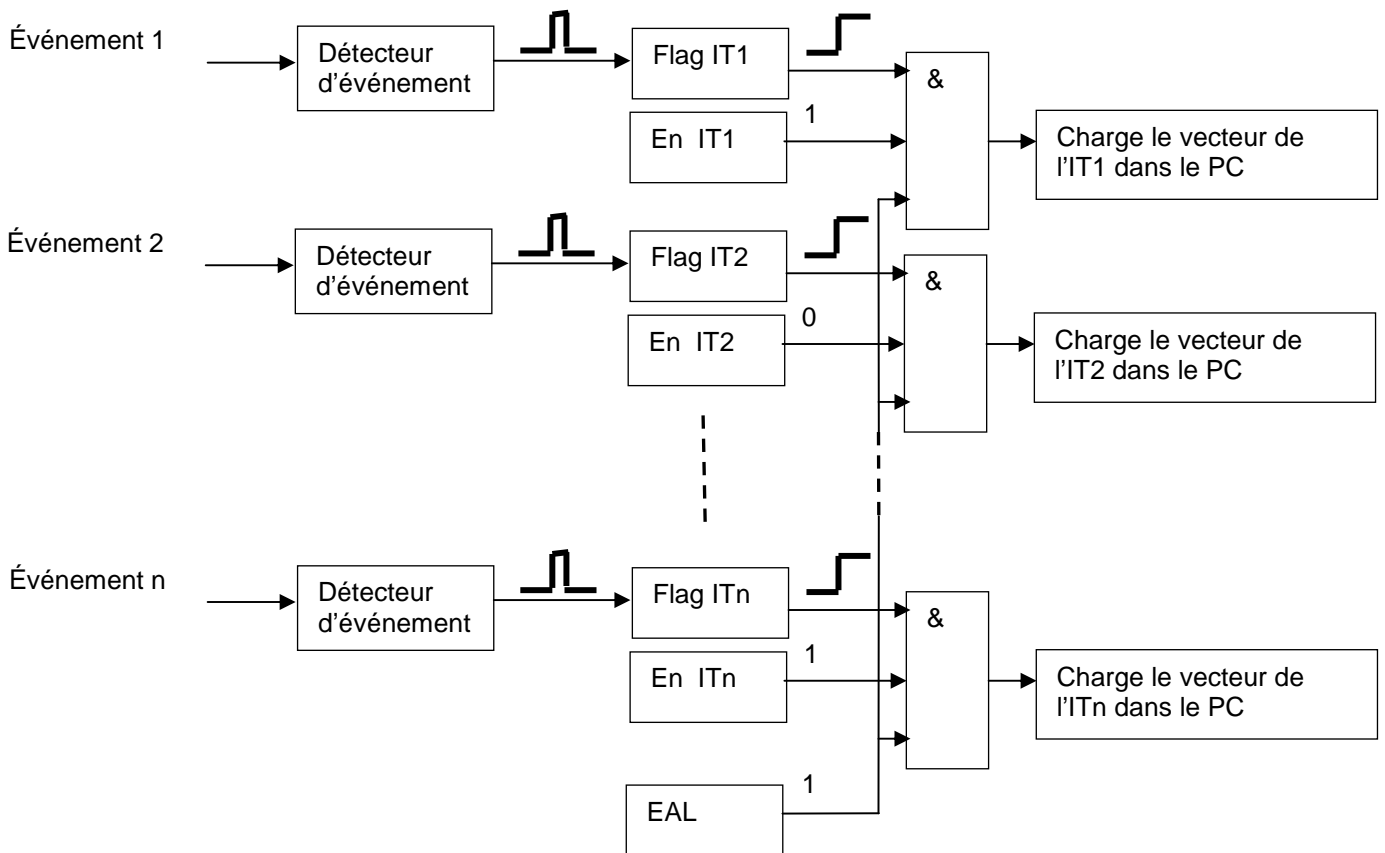
```

; SP d'interruption
    push  acc    ; sauvegarde du contexte
    push  psw
    push  12
...
    pop   12    ; ici est traitée la tâche en IT
    pop   psw
    pop   acc
    reti      ; retour de sous-programme d'IT
    
```

12.2. Masquage et validation

Par défaut la plupart des sources d'interruptions sont désactivée (masquée) lors du RESET. Certaines sources ne sont pas masquables (NMI : Non Masquable Interrupt)

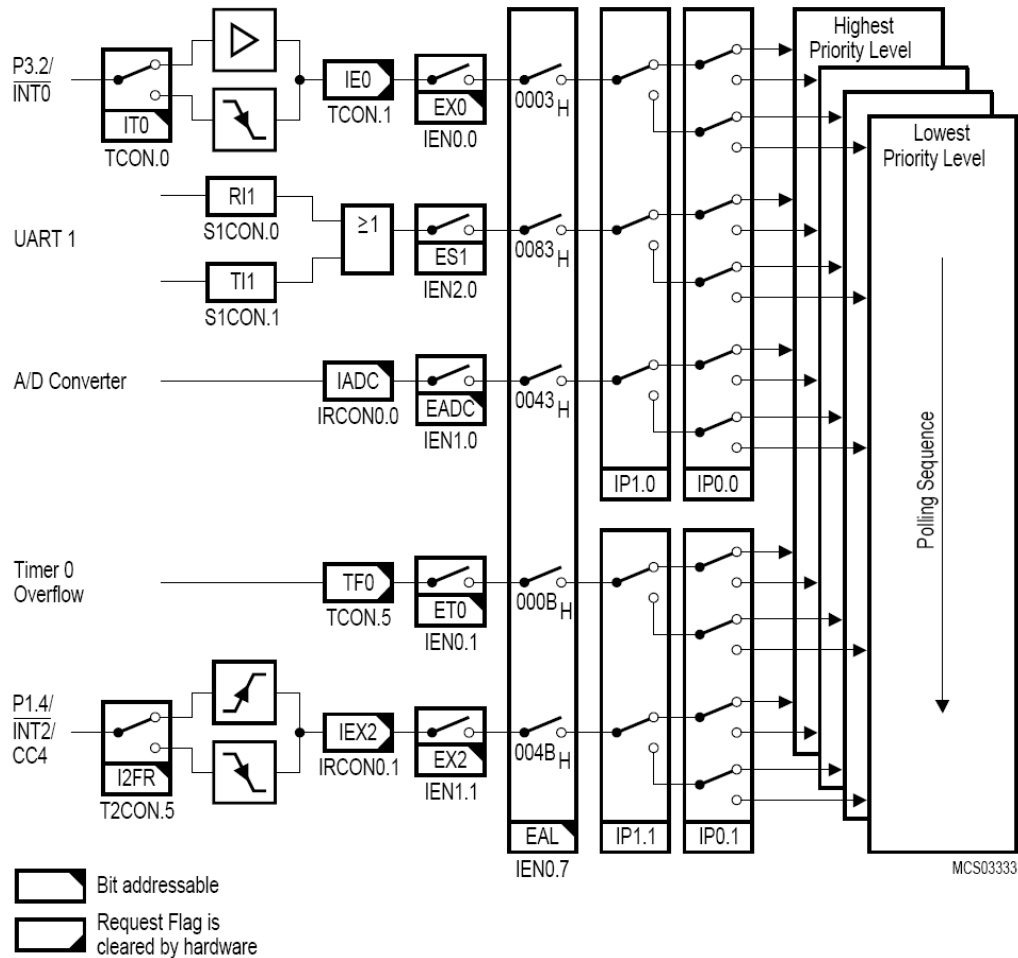
Masquage et validation des interruptions, exemple :



Les événements détectés provoque la mise à '1' d'un drapeau (flag) qui leur est propre. Si le bit d'autorisation est à '1' et que le masque global (EAL) l'est aussi l'interruption a lieu : le PC est chargé avec l'adresse du vecteur.

Avant l'instruction de retour d'interruption (RETI) le masque DOIT être effacé par le programme.

Exemple de structure d'interruption (C517A)



Exemple de registre de configuration (C517A)

Special Function Register IEN0 (Address A8H)

Reset Value : 00H

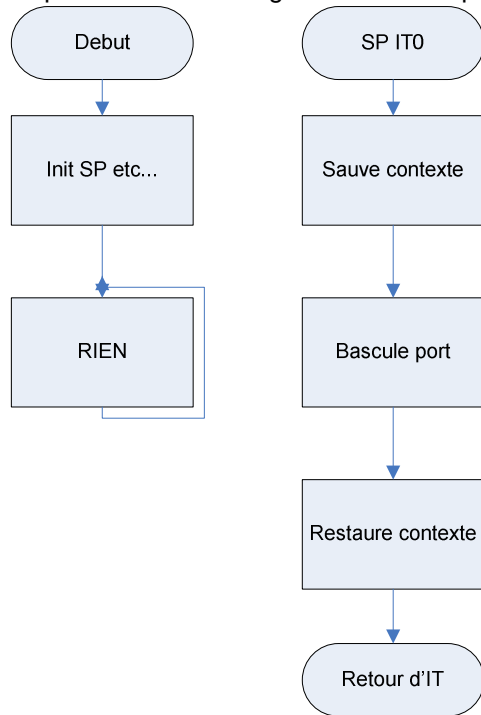
Bit No.	MSB							LSB	IEN0
	AF _H	AE _H	AD _H	AC _H	AB _H	AA _H	A9 _H	A8 _H	
A8 _H	EAL	WDT	ET2	ES0	ET1	EX1	ET0	EX0	

The shaded bit is not used for interrupt control

Bit	Function
EAL	Enable/disable all interrupts. If EA=0, no interrupt will be acknowledged. If EA=1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
ET2	Timer 2 interrupt enable. If ET2 = 0, the timer 2 interrupt is disabled.
ES0	Serial channel 0 interrupt enable If ES0 = 0, the serial channel interrupt 0 is disabled.
ET1	Timer 1 overflow interrupt enable. If ET1 = 0, the timer 1 interrupt is disabled.
EX1	External interrupt 1 enable. If EX1 = 0, the external interrupt 1 is disabled.
ET0	Timer 0 overflow interrupt enable. If ET0 = 0, the timer 0 interrupt is disabled.
EX0	External interrupt 0 enable. If EX0 = 0, the external interrupt 0 is disabled.

12.3. Exemple de mise en œuvre des interruptions

Le port P2.0 doit changer d'état à chaque front descendant sur P3.2 (/INT0).



```

CSEG      AT      0
ljmp      prog

ORG       3       ; adresse vecteur INT0
push     acc     ; push et pop pour demo
push     psw
cpl      P2.0   ; bascule P2.0
clr      IE0    ; efface drapeau d'IT
pop      psw
pop      acc
reti

prog:     mov     sp,#7Fh
          setb   IT0 ; detection front desc
          setb   EX0 ; active IT0
          setb   EAL ;active les IT demasquées
          sjmp  $
          END
  
```

Sources et vecteurs d'interruption du C517A :

Interrupt Source	Interrupt Vector Address	Interrupt Request Flags
External Interrupt 0	0003H	IE0
Timer 0 Overflow	000BH	TF0
External Interrupt 1	0013H	IE1
Timer 1 Overflow	001BH	TF1
Serial Channel 0	0023H	RI0 / TI0
Timer 2 Overflow / Ext. Reload	002BH	TF2 / EXF2
A/D Converter	0043H	IADC
External Interrupt 2	004BH	IEX2
External Interrupt 3	0053H	IEX3
External Interrupt 4	005BH	IEX4
External Interrupt 5	0063H	IEX5
External Interrupt 6	006BH	IEX6
Serial Channel 1	0083H	RI1 / TI1
Compare Match Interrupt of Compare Registers CM0-CM7 assigned to Timer 2	0093H	ICMP0 - ICMP7
Compare Timer Overflow	009BH	CTF
Compare Match Interrupt of Compare Register COMSET	00A3H	ICS
Compare Match Interrupt of Compare Register COMCLR	00ABH	ICR

REMARQUE : Le vecteur de RESET se trouve à l'adresse 0x0000 et les vecteurs d'interruptions se trouvent aux adresses 0x0003 à 0x00AB.

Pour pouvoir utiliser ces vecteurs il est nécessaire de placer en 0x0000 un saut absolu (LJMP) sur une adresse hors de la zone des vecteurs d'interruption.

L'outil de développement KEIL propose un entête de programme contenant les déclarations des différents segments mémoire, une réservation de RAM pour la pile SP ainsi qu'un saut en 0x0000 sur le segment de programme.

entete.A51

```

NAME                module_name
;-----
; FONCTION:
;
;
;-----
; AUTEUR            :
; Développé le      :
; Dernière modif le :
;-----
#include <reg517A.inc> ; include définition des registres du µC Infineon C515C
;-----
; XDATA SEGMENT--Reserves space in external RAM--Delete this segment if not used.
;-----
xdata_seg_name SEGMENT XDATA          ; segment for XDATA RAM
                RSEG   xdata_seg_name ; switch to this xdata segment
xdata_variable: DS      1              ; reserve 1 Bytes for xdata_variable
;-----
; DATA SEGMENT--Reserves space in DATA RAM--Delete this segment if not used.
;-----
data_seg_name   SEGMENT DATA          ; segment for DATA RAM.
                RSEG   data_seg_name  ; switch to this data segment
data_variable:  DS      1              ; reserve 1 Bytes for data_variable
;-----
; IDATA SEGMENT--Reserves space in indirect DATA RAM--Delete this segment if not
used.
;-----
idata_seg_name  SEGMENT IDATA          ; segment for DATA RAM.
                RSEG   idata_seg_name ; switch to this data segment
idata_variable: DS      1              ; reserve 1 Bytes for data_variable
;-----
; BIT SEGMENT--Reserves space in BIT RAM--Delete segment if not used.
;-----
bit_seg_name    SEGMENT BIT            ; segment for BIT RAM.
                RSEG   bit_seg_name   ; switch to this bit segment
bit_variable:   DBIT    1              ; reserve 1 Bit for bit_variable
bit_variable1:  DBIT    4              ; reserve 4 Bits for bit_variable1
;-----
; Add constant (typeless) numbers here.
;-----
typeless_number EQU    0DH             ; assign 0D hex
typeless_num1   EQU    typeless_number-8 ; evaluate typeless_num1
;-----
; -- Debugging with Monitor-51 needs          -- NE PAS DETRUIRE
;-----
                DSEG   AT      0x23
RESERVE:        DS      3
;-----
; STACK SEGMENT--Reserves space for STACK          -- NE PAS DETRUIRE
;-----
STACK          SEGMENT IDATA          ; Segment pour la pile
                RSEG   STACK
                DS      32            ; Taille de la pile
  
```

```
-----  
; Provide an LJMP to start at the reset address (address 0) in the main module.  
; You may use this style for interrupt service routines.  
-----  
                CSEG    AT    0        ; absolute Segment at Address 0  
                LJMP   start        ; reset location (jump to start)  
-----  
; CODE SEGMENT--Reserves space in CODE ROM for assembler instructions.  
-----  
PROG            SEGMENT CODE  
                RSEG    PROG        ; switch to this code segment  
start:          MOV    SP,#STACK-1  
-----  
; Insert your assembly program here.  
-----  
  
-----  
; The END directive is ALWAYS required.  
-----  
                END                ; End Of File
```





13. TIMER

Les microcontrôleurs intègrent des compteurs (8 ou 16 bits) qui suivant leur utilisation sont nommé **TIMER** ou **COMPTEUR**.

Utilisation comme **TIMER** : généralement lorsque leur horloge est « constante » ils peuvent alors mesurer ou produire des « temps ».

Utilisation comme **COMPTEUR** : Généralement ils comptent des événements (fronts montants ou descendant) sur une entrée physique du microcontrôleur.

Le C517A possède trois **TIMER/COUNTER** avec des applications différentes pour chaque.

Les **TIMER/COUNTER** 0 et 1 peuvent être configure en 4 modes :

Mode 0: 8-bit timer/counter avec un pré-diviseur par 32

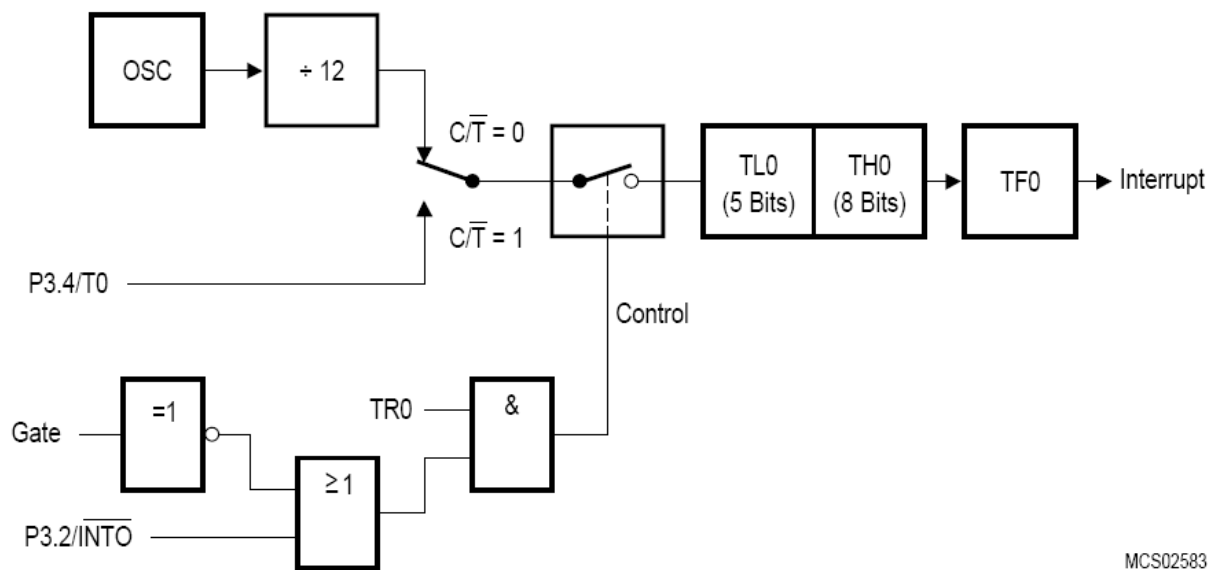
Mode 1: 16-bit timer/counter

Mode 2: 8-bit timer/counter avec rechargement automatique

Mode 3: Timer/counter 0 est un timer/counter 8-bit et Timer/counter 1 est un **TIMER** 8 bits.

Les entrées **INT0** et **INT1** peuvent être utilisées comme en entrée de validation (gate) pour ces deux **TIMER**, cela facilite la mesure de temps

MODE 0 pour **TIMER 0** (idem pour **TIMER 1**) :



Le bit **C/T** (Counter/Timer) permet de choisir la source de l'horloge, **OSC/12** ou la broche **T0** le comptage s'effectue sur les 8 bits de **TH0** (**TL0** servant de pré-diviseur par $32 = 2^5$)

TR0 (**TIMER RUN 0**) active le **TIMER** si :

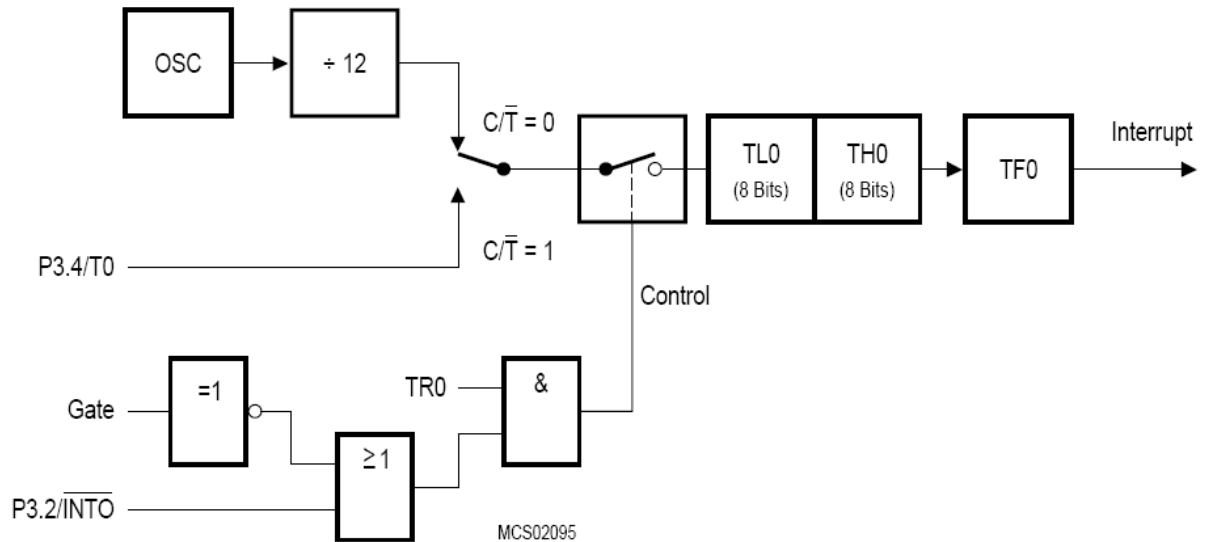
Le bit **GATE** est à 0

Ou Le bit **Gate** est 1 ET la broche **/INT0** est à 1

Lorsqu'il y a débordement (passage de **TF0** de 0xFF à 0x00), le bit **TF0** passe à 1, une interruption peut être générée (**TF0** DOIT être remis à 0 par programme).

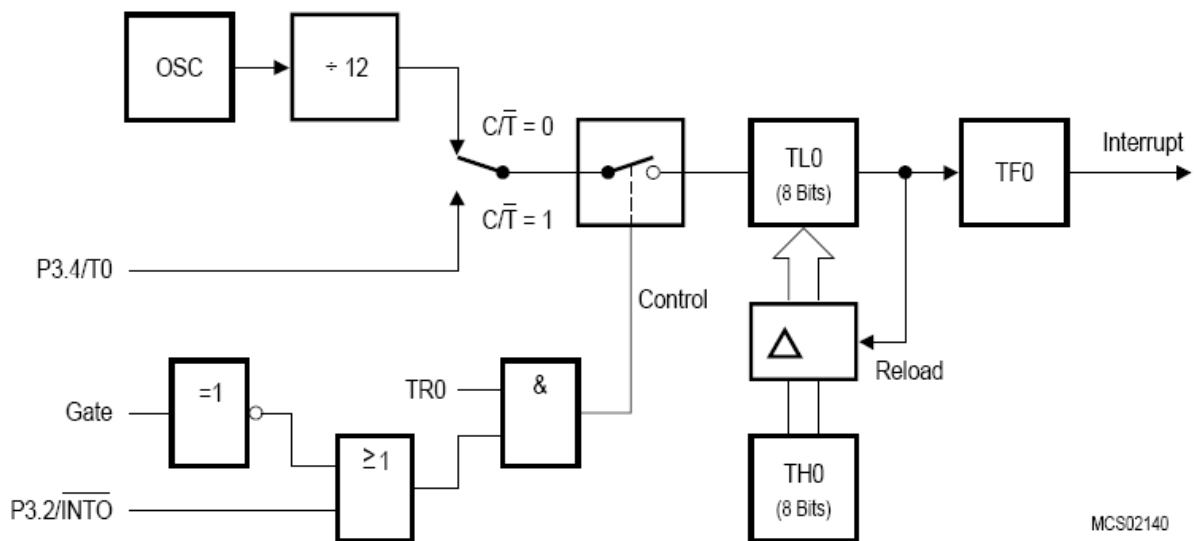
L'interruption est activée si **ET0** du registre **IEN0** est à 1.

MODE 1 pour TIMER 0 (idem pour TIMER 1)



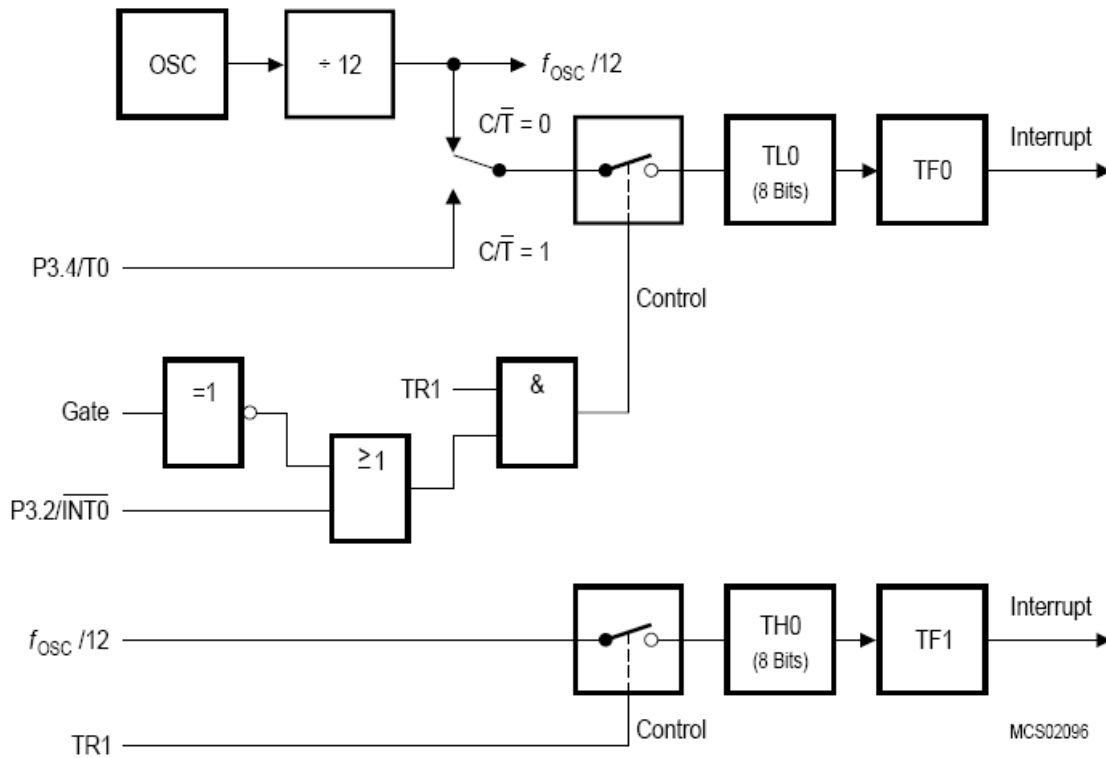
Le fonctionnement est identique au mode 0. Le comptage se fait sur 16 bits et il n'y a pas de pré-diviseur.

MODE 2 pour TIMER 0 (idem pour TIMER 1)



Lors du débordement (overflow) le registre TL0 du TIMER/COUNTER est automatiquement rechargé avec le contenu du registre TH0. La durée avec débordement est donc $0xFFFF - TH0$. Ce mode est particulièrement adapté pour générer des signaux rectangulaires. Dans le cas d'un signal carré il n'y a qu'à basculer le port de sortie, dans le cas un rectangle il faut dans l'interruption recharger TH0 avec la prochaine valeur.

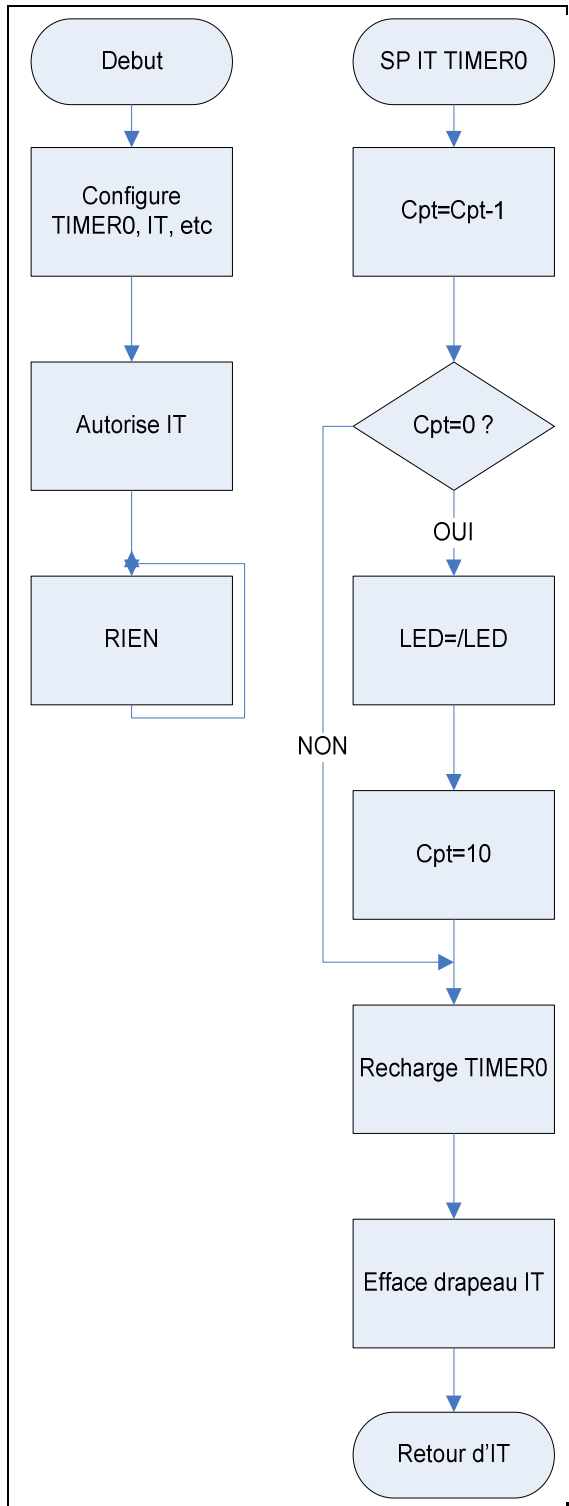
MODE 3



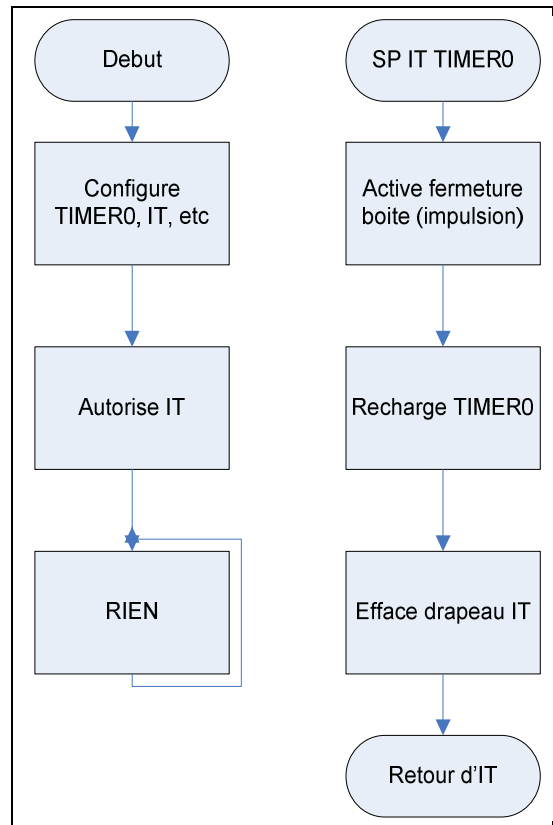
Le TIMER/COUNTER 0 compte sur 8 bits, le TIMER1 utilise le registre TH0 comme compteur (8 bits).

Exemple : générer un signal carree sur P2.0 (T=0,5S)

```
#include <reg517A.inc>
data_seg_name SEGMENT DATA ; segment for DATA RAM.
                RSEG data_seg_name ; switch to this data segment
cpt:            DS 1 ; compteur d'interruptions
;
valTH0 EQU 3Ch ; 3CB0 = FFFFh-50000d+1 (0-50000)
valTL0 EQU 0B0h
LED EQU P2.0
;
STACK SEGMENT IDATA ; Segment pour la pile
RSEG STACK
DS 32 ; Taille de la pile
;
CSEG AT 0 ; absolute Segment at Address 0
LJMP start ; reset location (jump to start)
ORG 0x000B ; vecteur TIMER0 ovf
LJMP IT_TIMER0
;
PROG SEGMENT CODE
RSEG PROG ; switch to this code segment
start:
MOV SP,#STACK-1
MOV TMOD,#00000001b ; TIMER 0 sur mode 1 source Q/12
SETB TR0 ; run TIMER 0
MOV CPT,#10 ; compteur d'IT
SETB ET0 ; autorise IT sur debordement TIMER0
SETB EAL ; autorise toutes les IT
SJMP $ ; attend IT
;
IT_TIMER0: DJNZ CPT,NON
CPL LED ; bascule LED
MOV CPT,#10 ; recharge le compteur d'IT
NON: MOV TH0,#valTH0 ; recharge TIMER0
MOV TL0,#valTL0
CLR TF0 ; efface drapeau IT0
RETI
END ; End Of File
```



EXEMPLE 1



EXEMPLE 2

Exemple 2 : Comptage d'événement. Un capteur optique détecte le passage de bonbons sur un tapis roulant les entraînant dans une boîte. La boîte doit être fermée lorsqu'elle contient dix bonbons.

```

#include <reg517A.inc>
data_seg_name SEGMENT DATA ; segment for DATA RAM.
                RSEG data_seg_name ; switch to this data segment
CPT:           DS 1 ; compteur de boites
;
valTH0 EQU 0FFh ; 10 bonbons par boite
valTL0 EQU 0F6h
BOITE EQU P2.0
STACK SEGMENT IDATA ; Segment pour la pile
        RSEG STACK
        DS 32 ; Taille de la pile
;
        CSEG AT 0 ; absolute Segment at Address 0
        LJMP start ; reset location (jump to start)
        ORG 0x000B ; vecteur TIMER0 ovf
        LJMP IT_TIMER0
;
PROG SEGMENT CODE
        RSEG PROG ; switch to this code segment
start: MOV SP,#STACK-1
        MOV TMOD,#00000101b ; TIMER 0 sur mode 1 source Q/12
        CLR A ; compteur de boites
        MOV CPT,A
        MOV TH0,#valTH0 ; recharge TIMER0
        MOV TL0,#valTL0
        SETB ET0 ; autorise IT sur debordement TIMER0
        SETB EAL ; autorise toutes les IT
        SETB TR0 ; run TIMER 0
        SJMP $ ; attend IT
;
IT_TIMER0: SETB BOITE ; impulsion sur fermeture
           CLR BOITE
           INC CPT ;incrémente compteur
           MOV TH0,#valTH0 ; recharge TIMER0
           MOV TL0,#valTL0
           CLR TF0 ; efface drapeau IT0
           RETI
           END ; End Of File
  
```

Essayer les exemples ci-dessus dans le simulateur, on constate une petite erreur sur la durée de la période. En effet, les TIMER sont rechargés par logiciel, le décalage correspond aux instructions d'appel de sous-programme d'interruption et de rechargement des registres du TIMER

14. Le TIMER 2

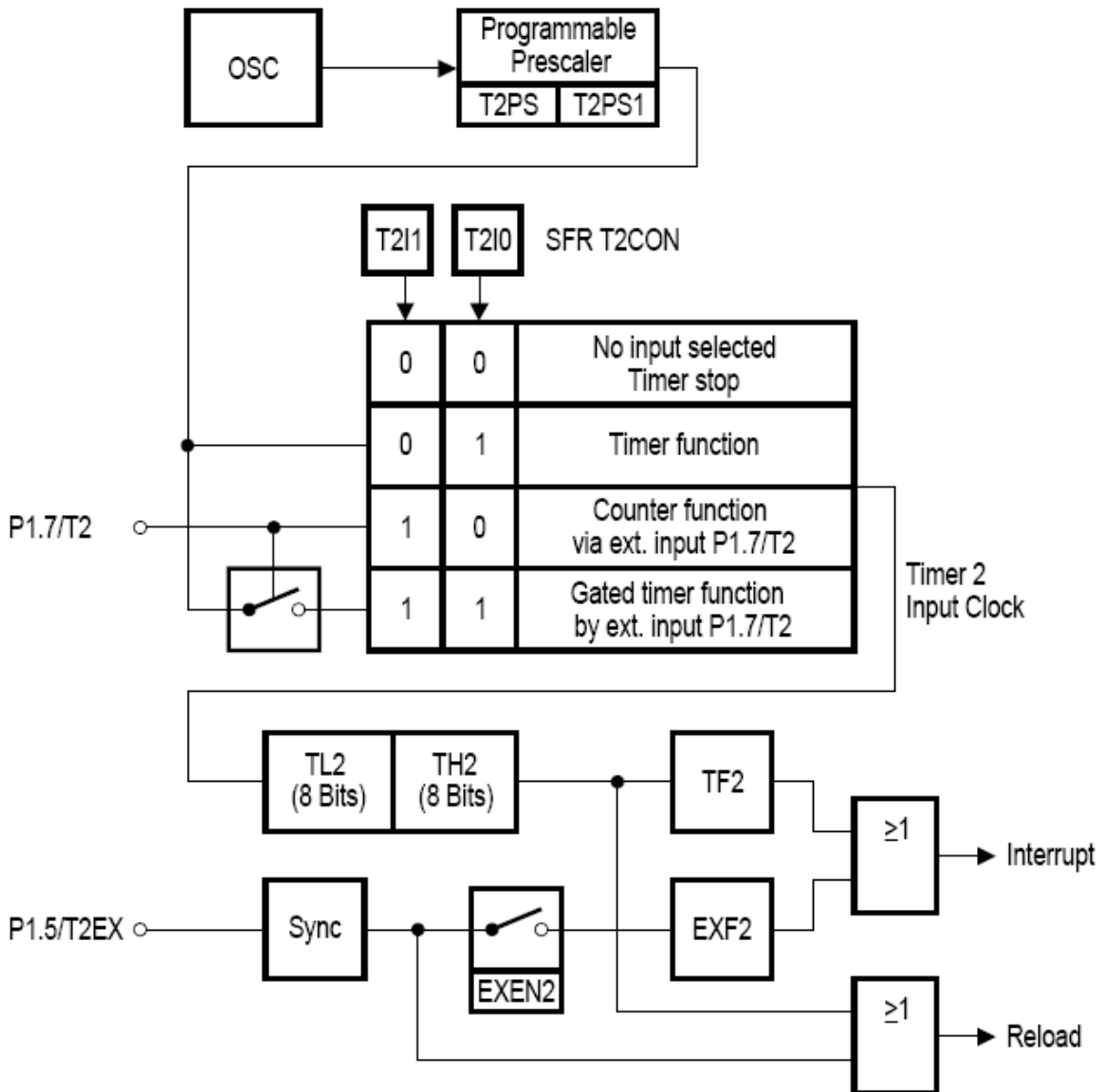
Les TIMER nécessitent un apport logiciel afin d'actionner les sorties. Les fonctions matérielles CAPTURE/COMPARE permettent de limiter considérablement le logiciel.

COMPARE automatisent les actions sur les broches externes.

CAPTURE facilite l'échantillonnage temporel pour la mesure de durée.

CAPTURE/COMPARE permet une réaction immédiate aux événements (ce n'est plus le logiciel qui recharge le TIMER mais un signal agissant directement sur l'entrée des registres, la précision temporelle est alors celle de l'oscillateur.

Les fonctions CAPTURE/COMPARE s'appuient sur le TIMER 2 ou le « COMPARE TIMER »
Modes opératoires du TIMER 2



T211, T210 = 0,1 le TIMER 2 compte l'horloge issue du prédiviseur OSC/12 /24 /48 /96

T211, T210 = 1,0 le COUNTER 2 compte les fronts descendants sur T2

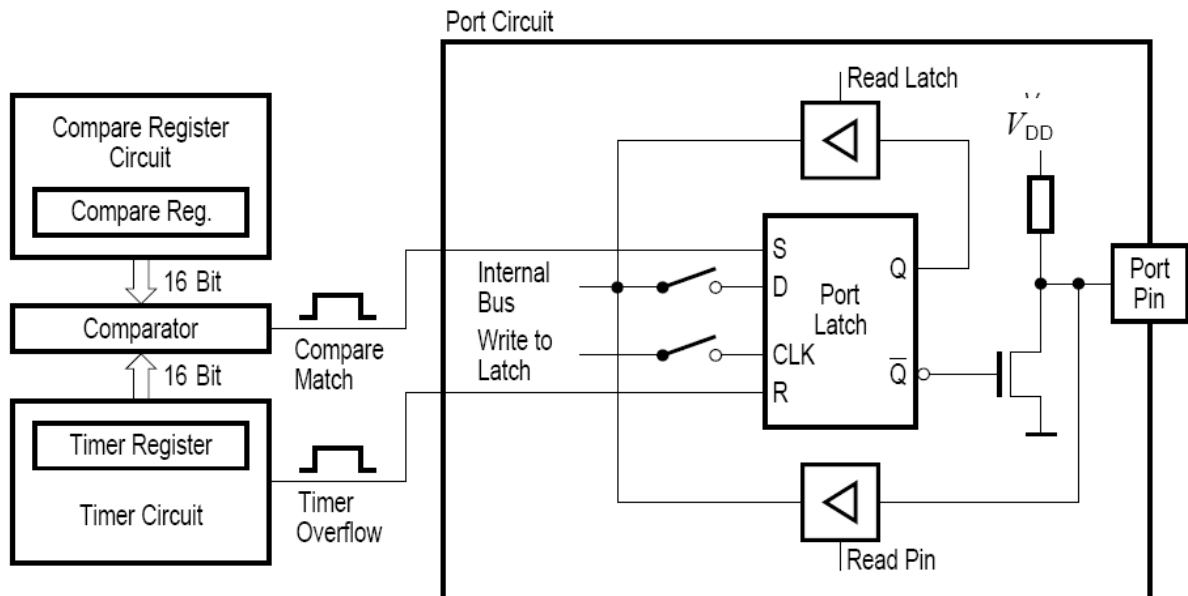
T211, 0T210 = 1,1 le TIMER 2 compte l'horloge issue du prédiviseur OSC/12 (uniquement) lorsque T2=1

En cas de débordement TF2=1 une interruption peut être activée.

15. COMPARE sur C517A

COMPARE AVEC TIMER 2 (fonctions CCx)

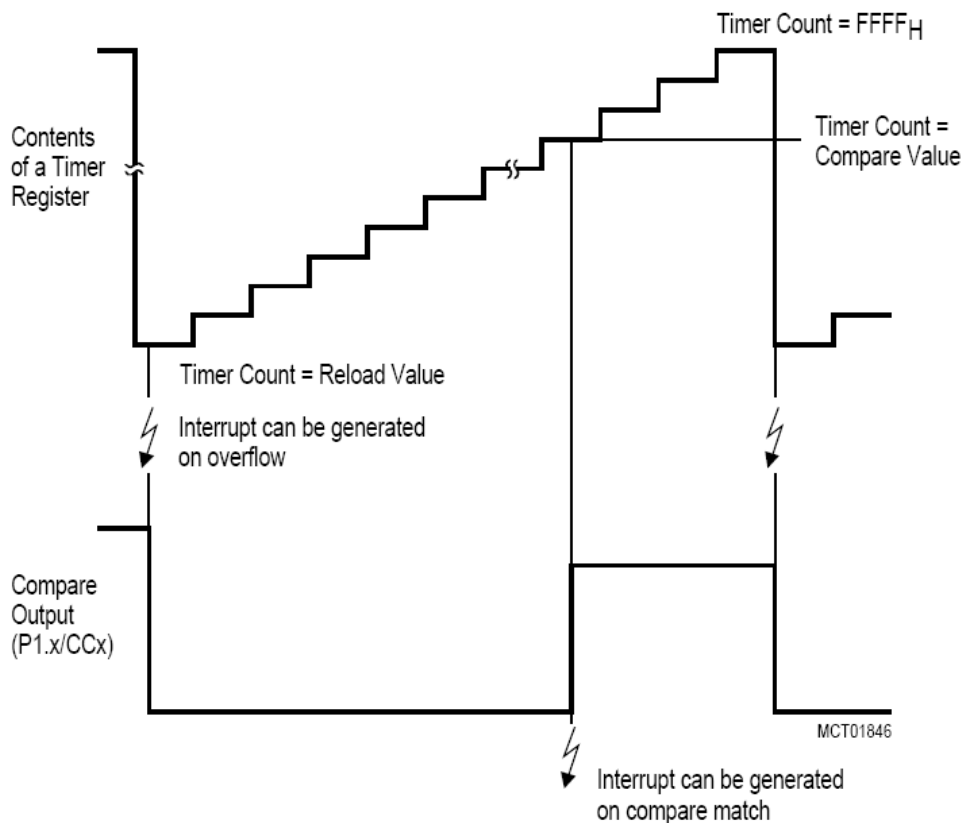
MODE 0



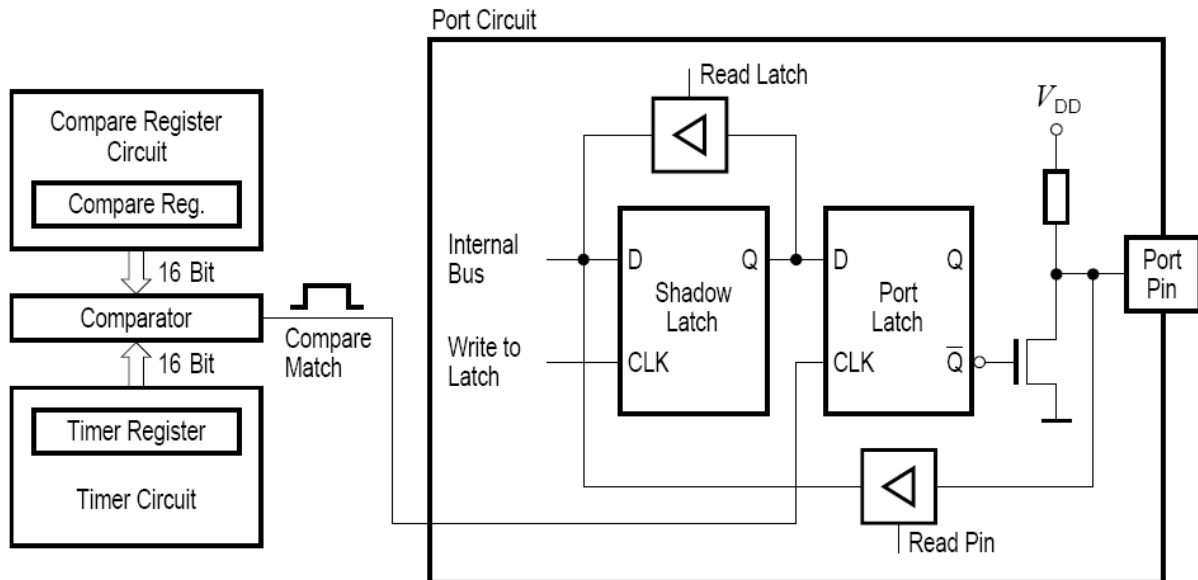
Lorsqu'il y a coïncidence (match) entre le TIMER 2 et le registre de comparaison (CCx) la broche sélectionnée est mise à 1. (S=1 \Rightarrow Q=1 et /Q=0 \Rightarrow le transistor est bloqué \Rightarrow le pull-up ramène un NL1 sur la sortie)

Lorsque le TIMER déborde la broche sélectionnée est mise à 0 et le TIMER peut être rechargé.

Ce principe est particulièrement adapté à la modulation de largeur d'impulsion, Pulse Wave Modulation (PWM) pour le contrôle de vitesse des moteurs ou la synthèse vocale .

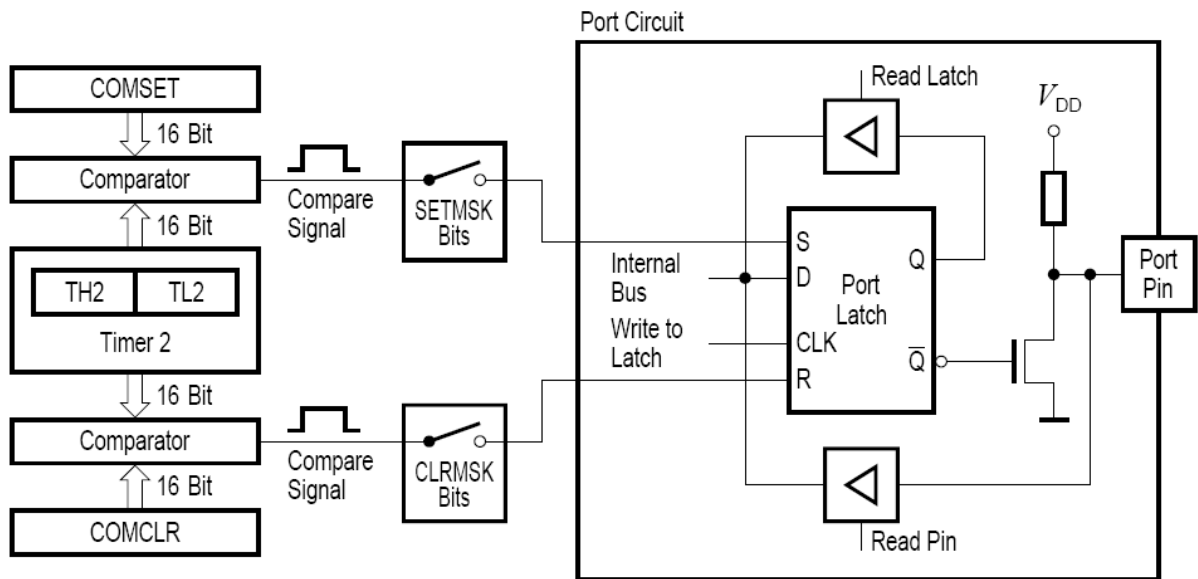


MODE 1



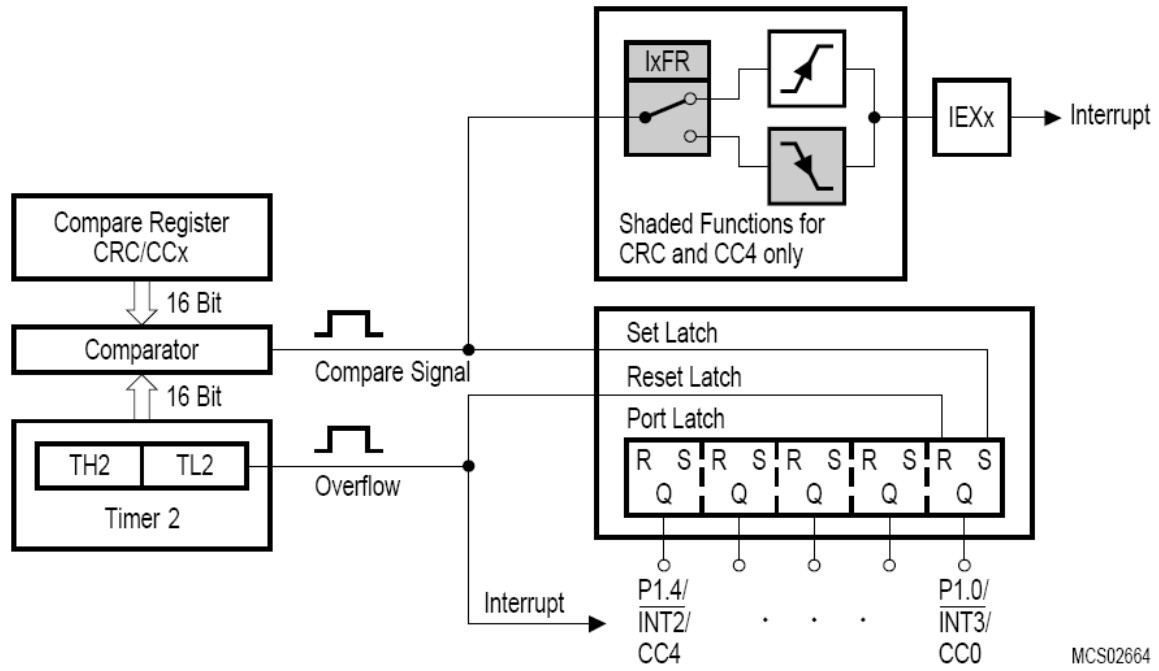
A chaque coïncidence la donnée dans le Latch de sortie est recopiée sur le port. Cette donnée est choisie par programmation et placée sur le Latch interne. Il donc est possible de choisir le niveau de sortie lors de la coïncidence

MODE 2

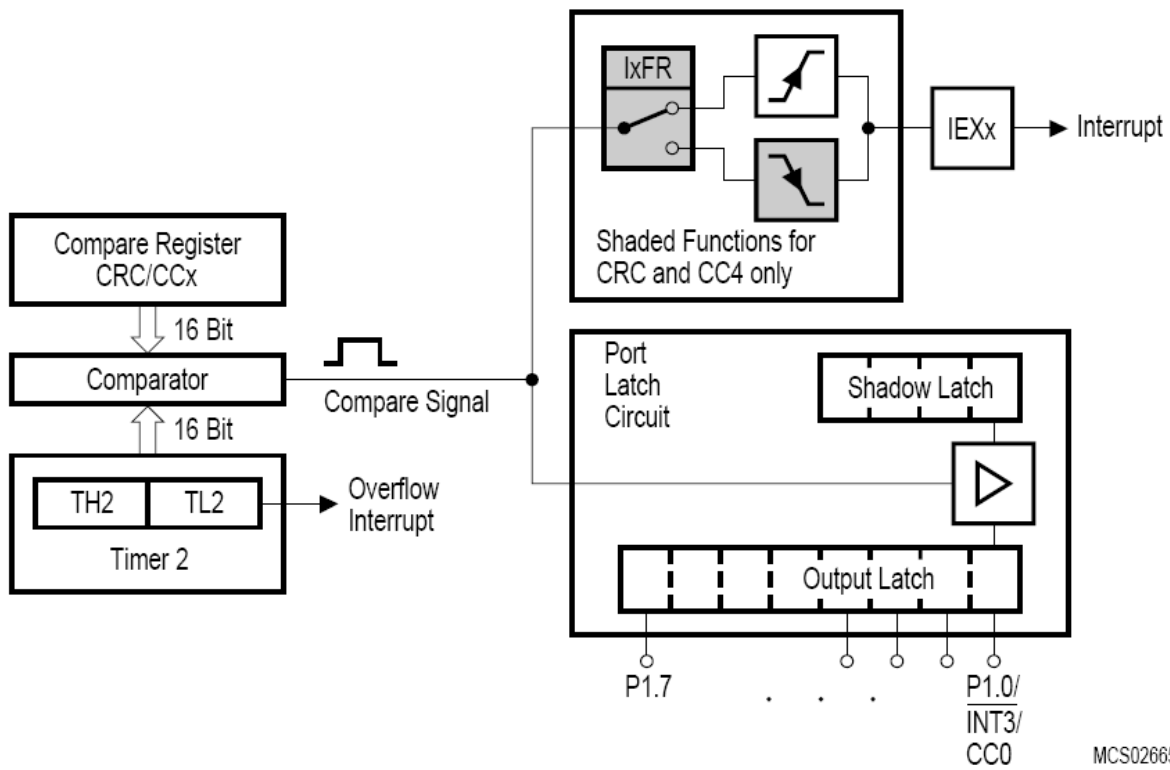


Quand il y a coïncidence entre le TIMER 2 et le registre COMSET un NL1 apparaît sur le port en sortie si le bit de masque SETMSK est à 1
Quand il y a coïncidence entre le TIMER 2 et le registre COMCLR un NL0 apparaît sur le port en sortie si le bit de masque CLRMSK est à 1

Commande simultanée de plusieurs broches en MODE 0 et 1:



MCS02664

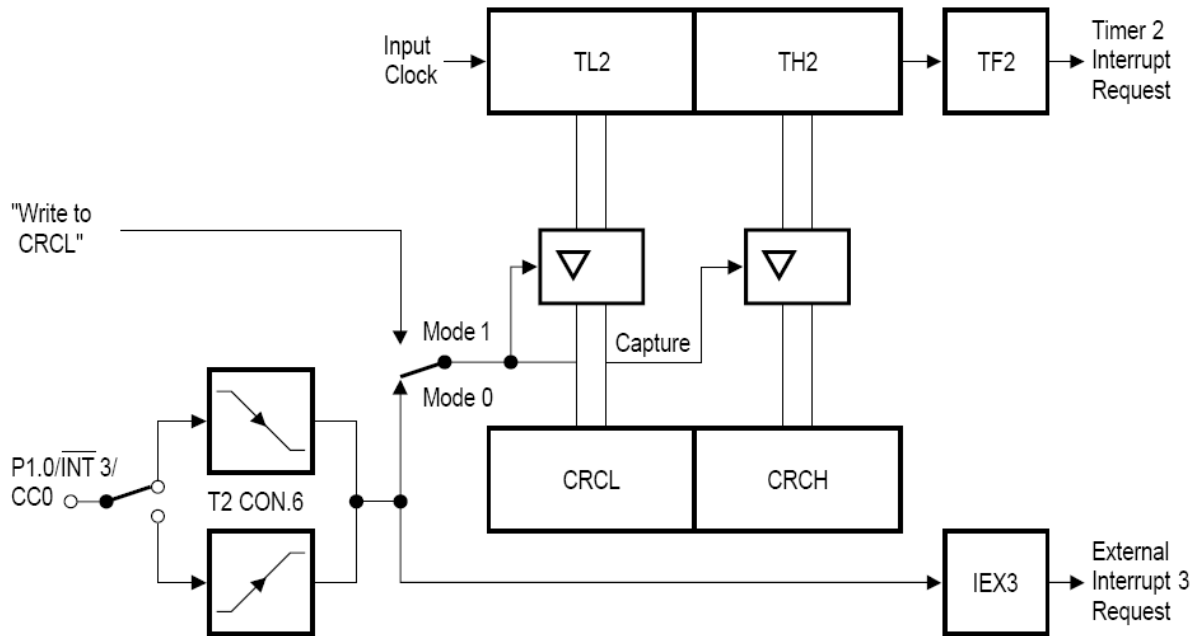


MCS02665

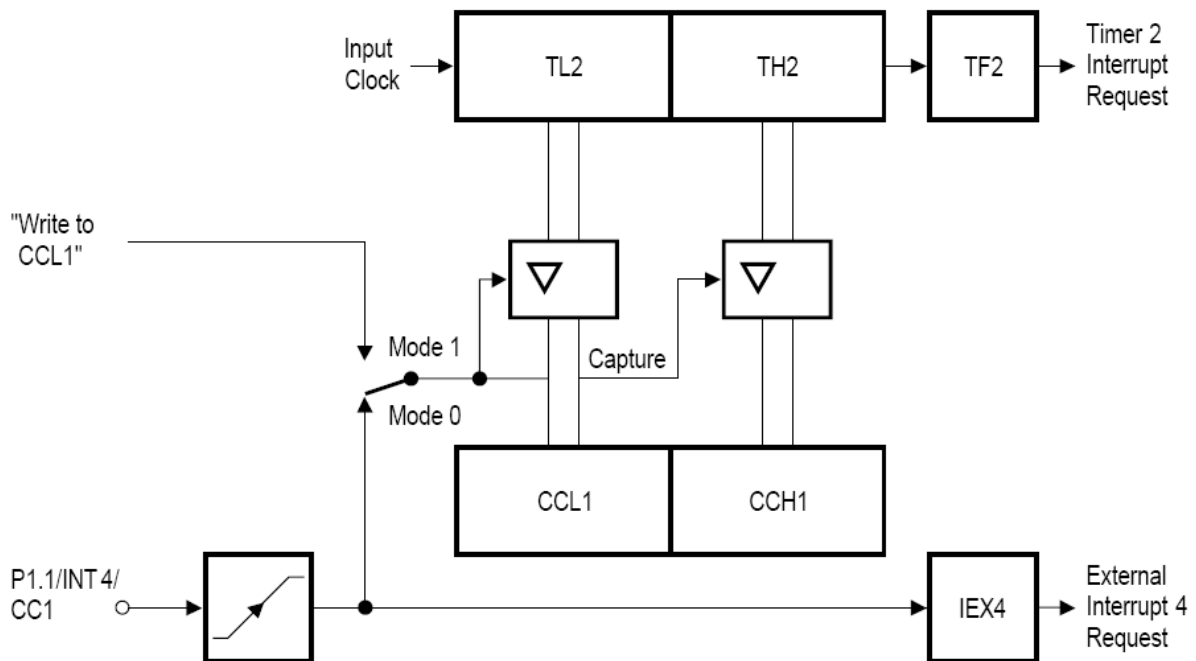
16. CAPTURE

Lors de l'événement (front montant ou descendant) sur une broche le contenu du TIMER 2 est recopié dans un registre, une interruption peut être générée. Le microcontrôleur conserve ainsi une trace très précise de l'instant où s'est produit l'événement. Le traitement et les actions correspondantes peuvent être traitées ensuite.

Avec les registres **CRC**, **CC4**



Avec les registres **CC1 to CC3**



17. ADC sur C517A

Le C517A possède un convertisseur analogique numérique 10 bits à approximations successives (ADC, Analog to Digital, Converter) à 12 entrées multiplexées. Une entrée devant être sélectionnée avant chaque conversion.

Il peut effectuer des conversions uniques ou continues. Un échantillonneur-bloqueur interne bloque le signal durant la conversion. La conversion peut être demandée (Start Of Conversion) par logiciel (écriture dans ADDTL) ou par un signal externe. La fin de conversion est signalée par un bit qui peut déclencher une interruption. Le résultat est placée dans deux registres 8 bits ADDATH et ADDATL. (seuls 2 bits de ADDATL sont valides, le résultat étant sur 10 bits)

Les références de tensions (Vref+, Vref-) sont placées sur deux broches externes.

Voir le schéma page 63

Exemple de gestion de l'ADC :

```

; Simple programme effectuant une conversion sur AN0 (P7.0)

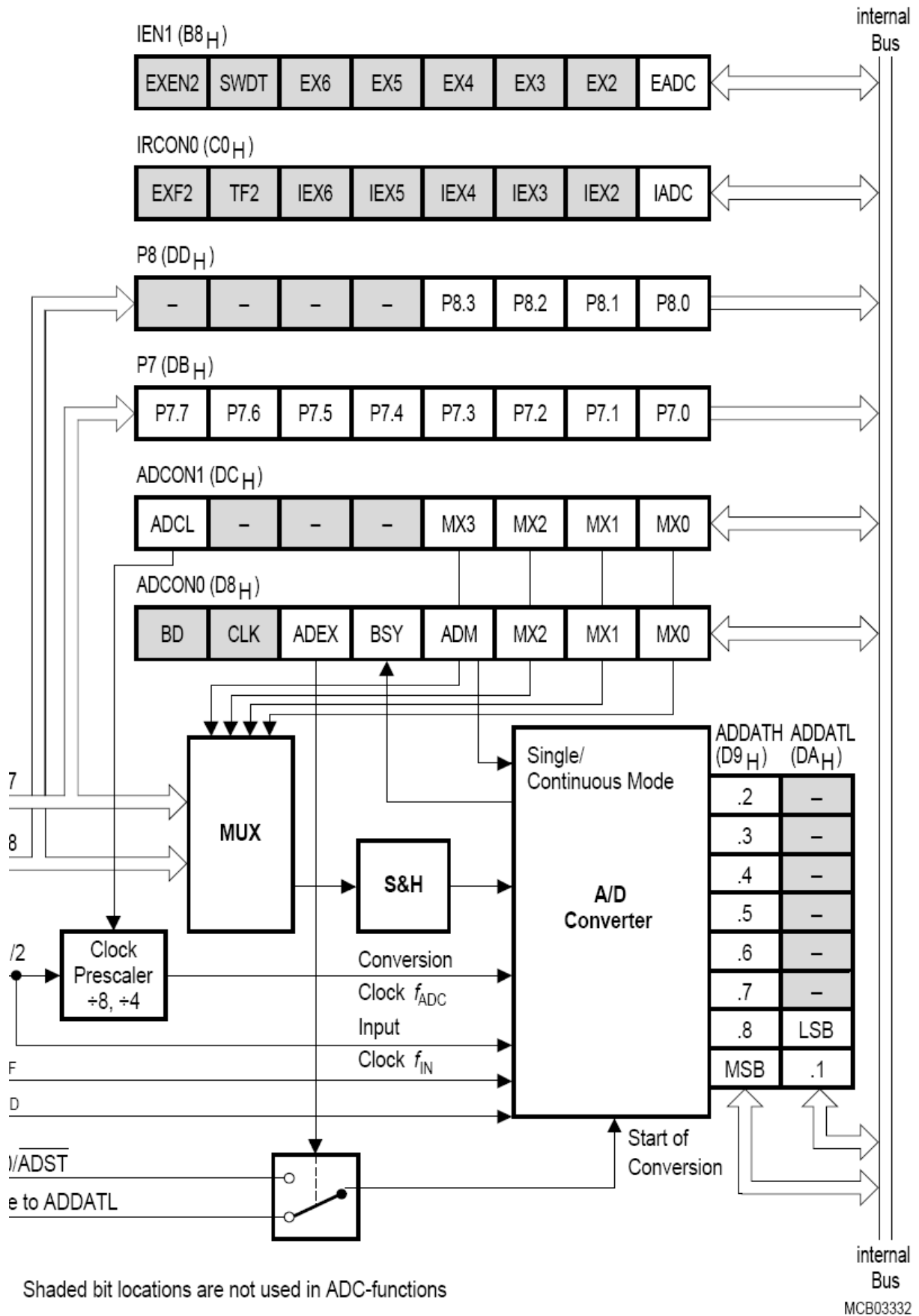
#include <reg517A.inc>
;
data_seg_name    SEGMENT DATA                ; segment for DATA RAM.
                  RSEG    data_seg_name      ; switch to this data segment
RESH:            DS    1                      ; poids forts du résultat
RESL:            DS    1                      ; poids faibles du résultat
;
                  CSEG    AT    0            ; absolute Segment at Address 0
                  LJMP   start              ; reset location (jump to start)
;
PROG             SEGMENT CODE
                  RSEG    PROG              ; switch to this code segment
MOV    ADCON0,#00000000b ; SOC interne, BSY=0, conversion unique
                           ; AN0 sélectionnée
CLR    EADC              ; pas d'interruption
MOV    ADDATL,#0         ; SOC
JB     BSY,$             ; attend EOC
CLR    BSY               ; rq : ces deux instructions sont
                           ; remplaçables par JBC
MOV    RESH,ADDATH       ; sauve le résultat
MOV    RESL,ADDATL
SJMP  $                  ; boucle sans fin
END                      ; End Of File
  
```

Exercice :

Afin de minimiser l'influence du bruit de quantification on souhaite effectuer la moyenne de 20 mesures avant de stocker le résultat.

Réaliser un sous-programme recevant dans ACCA le numéro du canal d'entrée puis effectuant 20 mesures et calculant leur moyenne avant stockage dans RESH, RESL. RESH contient les deux poids forts et RESL les huit poids faibles.

Réaliser un programme testant ce sous programme.

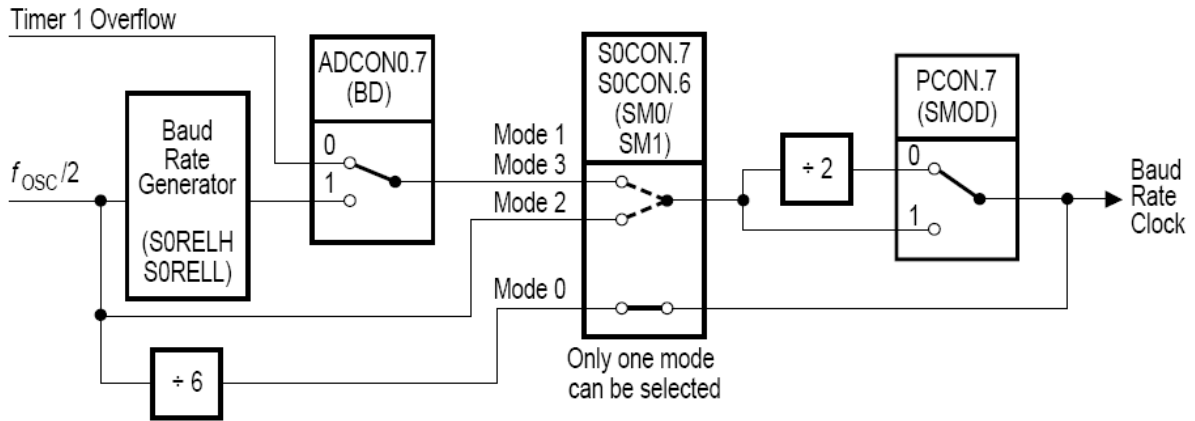


MCB03332

18. Communications séries

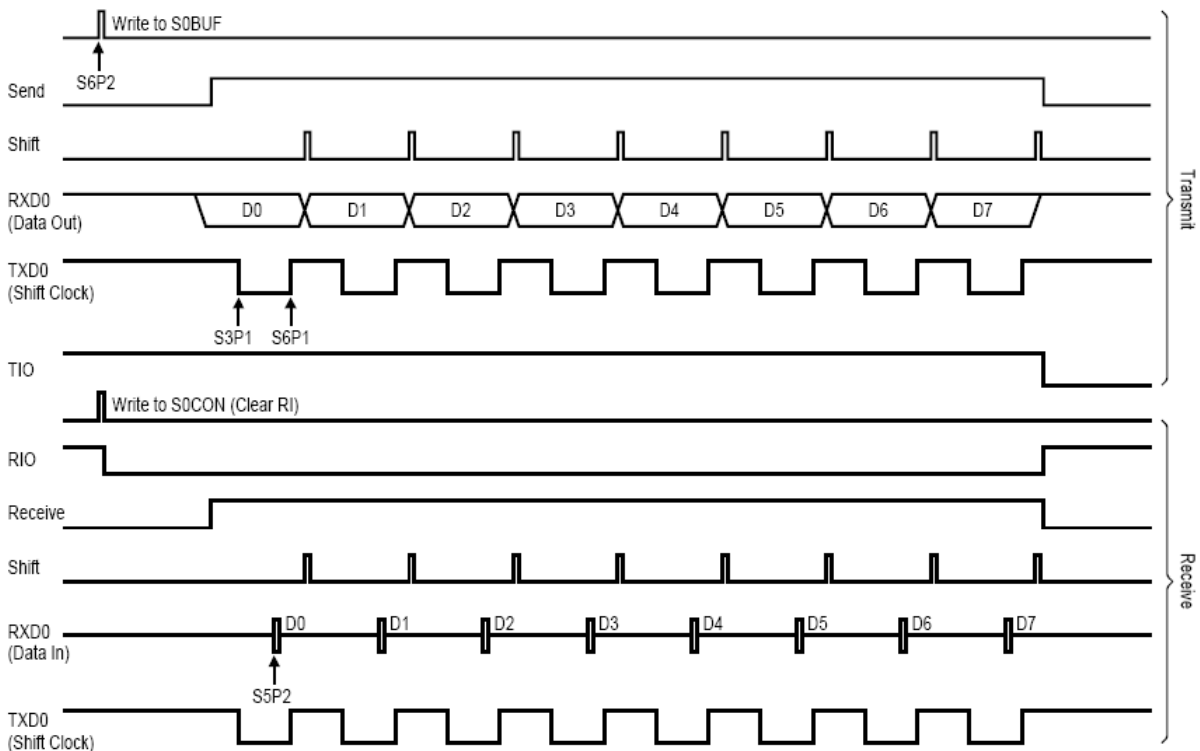
L'interface de communications peut gérer les communications synchrones (type SPI), mode maître (génère l'horloge) ou esclave (reçoit l'horloge) ainsi que les communications asynchrones (code NRZ)

L'horloge asynchrone (Baud Rate Clock) peut être générée à partir d'OSC/2 ou du TIMER1., les différents modes permettent d'obtenir de nombreuses fréquences de cadencement (le générateur de Baud est également utilisé pour les communications synchrones).



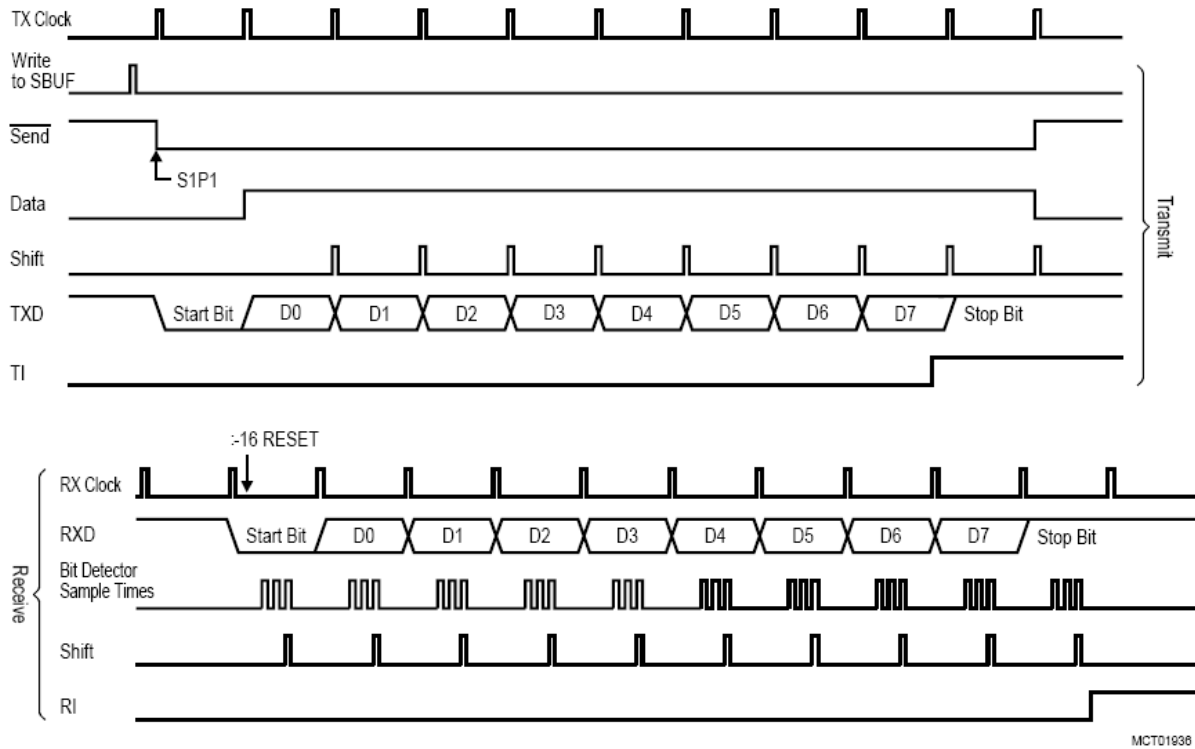
En mode synchrone les données sont transmises via un simple registre à décalage, la phase de l'horloge peut être ajustée de manière à synchroniser tous les types de récepteurs (fronts montants, dépendants etc.)

Mode synchrone

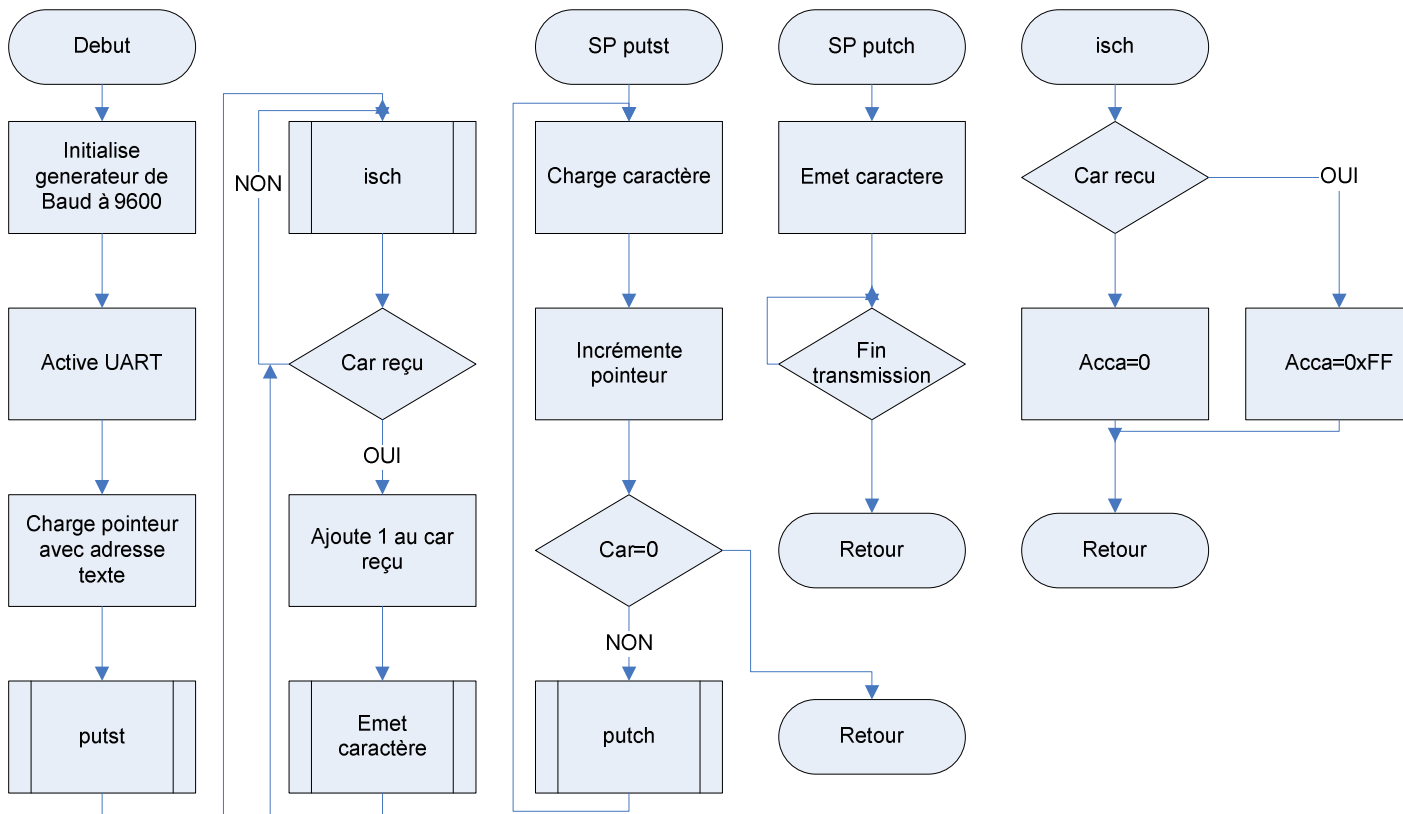


Une transmission est initiée par une écriture dans SxBUF. Les données sont alors émises et reçues simultanément de manière synchrone. Après 8 tops d'horloge SxBUF contient la donnée reçue. Les bits TI/RI indiquent la fin de la transmission/réception et peuvent générer une interruption, ces bits doivent être remis à zéro par logiciel.

Mode asynchrone



Send est un signal interne à l'USART qui valide la sortie des données
 Shift est un signal interne commandant le décalage du registre de transmission ou réception
 Une transmission est initiée par une écriture dans SxBUF. A la fin le bit TI passe à 1 et peut générer une interruption, ce bit doit être remis à zéro par logiciel.
 Une réception est initiée par la réception d'un bit de START. A la fin de la réception le bit RI passe à 1 et peut générer une interruption, ce bit doit être remis à zéro par logiciel. L'octet reçu se trouve dans SxBUF.



Exemple asynchrone: programme « écho », ce programme (simulable sur le port série 1), retourne en écho le caractère reçu.) voir page 27 « principe des communications asynchrones »

```

#include <reg517.inc>
sorelh    equ    0BAh    ; registres baud
sorell    equ    0AAh
TIO       equ    S0CON.1    ; bit fin reception
RIO       equ    S0CON.0    ; bit fin reception
stack     segment      IDATA ; pour pile S
prog      segment      code  ; zone programme
pconst    segment      code  ; pour constantes
          RSEG  stack
          DS    10h    ; reserve 16 octets pour la pile
;
          CSEG  AT 0    ; RAZ
          ljmp  start
;
          RSEG  prog
start:     mov    sp,#stack-1
          setb  adcon0.7    ; FOSC = baud generator
          mov   sorelh,#3    ;9600 Bauds (Q=24MHz)
          mov   sorell,#0D9h
          mov   S0CON,#01010000B ; mode 1, rx actif
          ljmp  tache
; SP emission d'un message (fin par 0)
putst:    mov   R0,#0FFh    ; R0 est le pointeur de caractère
suite:    inc   R0          ; caractère suivant
          mov   a,R0        ; dans acc
          movc  a,@a+dptr
          jz    fin         ; si acc=0 c'est finit
          call  putch       ; sinon émission caractère
          sjmp  suite      ; au suivant !
fin:      ret
; SP emission d'un caractère (dans acc)
putch:    mov   S0BUF,a      ; caractère dans registre de
transmission
          jnb   TIO,$       ; transmission terminée ?
          clr   TIO        ; efface drapeau
          ret
; SP test caractere reçu
isch:     jnb   RIO,non     ; attend caractère
          mov   a,#0FFh    ; si oui acc=FF
          ret
non:      clr   a          ; si non acc=0
          ret
; SP reception caractere
getch:    clr   RIO        ; efface drapeau
          mov   a,S0BUF
          ret
; pour test, emission du message puis echo+1
tache:    mov   dptr,#txt    ; adresse message dans pointeur ROM
          call  putst       ; emission message
repete:   call  isch       ; attend reception
          jz    repete
          call  getch       ; récupère le caractère reçu
          inc   a          ; retourne car +1
          call  putch
          sjmp  repete      ; boucle sans fin
; message à transmettre
          RSEG  PCONST
txt:      db    'Test communications - ECHO+1',10,13,0
          END
  
```