UNIVERSITE D'EVRY
VAL D'ESSONNE
EVRY
IUT
BRETIGNY-ATHIS MONS

GénieElectrique et InformatiqueIndustrielle
22 Allée Jean Rostand
91000 EVRY

01-69-47-72-20
secretariat-geii@iut.univ-evry.fr

µC 8051

P. Hoppenot
01 69 47 72 72
hoppenot@iup.univ-evry.fr
http://lsc.univ-evry.fr:8080/~hoppenot/

Avant propos

Ce cours est destiné aux étudiants de l'IUT GEII d'Evry dans le cadre de l'informatique industrielle. Il peut être utilisé librement dans la mesure où :

- ce n'est pas à des fins commerciales,
- la référence au présent document et à ses auteurs est clairement précisée.

Si vous avez des remarques pour enrichir ce document, n'hésitez pas à les transmettre à : hoppenot@iup.univ-evry.fr.

I.Présentation

I.1.Rappel sur les systèmes à microprocesseur

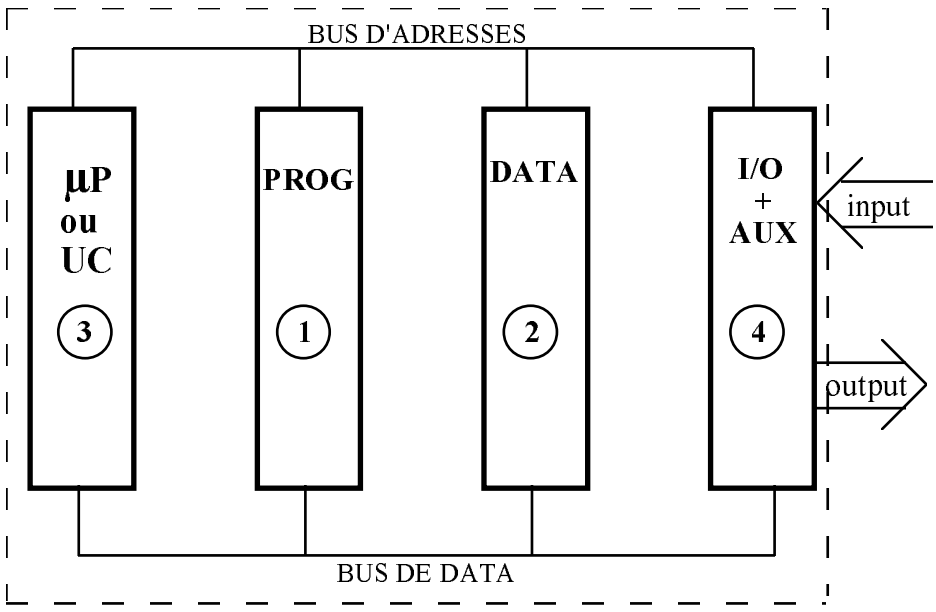


Fig. 1

Figure 1 : *Structure d'un système à microprocesseur.*

µP ou UC : Microprocesseur ou unité de calcul.
PROG† : Zone mémoire programme.
DATA : Zone mémoire de données.
I/O : Entrées sorties (Ports, liaison série…).
AUX : Auxiliaires (TIMER, gestionnaires d'interruptions, etc…).
1, 2, 3, 4 : Ordre de présentation des éléments.

I.2.Microcontrôleur 8051

Un microcontrôleur intègre toutes ces composantes ou au moins une partie. La mise en oeuvre est plus simple mais avec moins de ressources (dimension de la mémoire par exemple).

Exemples : 8031 : tout sauf zone PROG.
8051 : zone PROG = mémoire ROM (4 ko).
8751 : zone PROG = mémoire REPROM (4 ko).

Pour le 8031 il est nécessaire de se servir des PORTS d'entrées/sorties en tant que bus d'adresses et bus de données afin d'accéder à une mémoire programme extérieure au µC.

La mémoire programme du 8051 est une mémoire à fusibles; ceci implique un appareil de programmation capable de supporter la programmation de ce type de circuit. Par contre si une erreur a été commise lors de la programmation il n'est plus possible de la corriger.

La mémoire programme du 8751 est une mémoire reprogrammable; cela nous permet, en cas d'erreur, d'effacer le programme grâce à un effaceur à rayonnement ultraviolet et de reprogrammer le µC.

Un PORT correspond à 8 "pattes" du circuit sur lesquelles il est possible de recevoir ou d'envoyer des informations binaires sur chacune des pattes.

Pour les circuits 8051 et 8751 qui possèdent déjà une mémoire programme de 4K octets, il sera possible d'augmenter la taille de la mémoire programme par l'adjonction d'un boîtier externe (utilisation des PORTS d'entrées/sorties).

Pour tous les circuits il est possible d'augmenter la mémoire de données par l'adjonction d'une mémoire externe au microcontrôleur.

I.3.Brochage

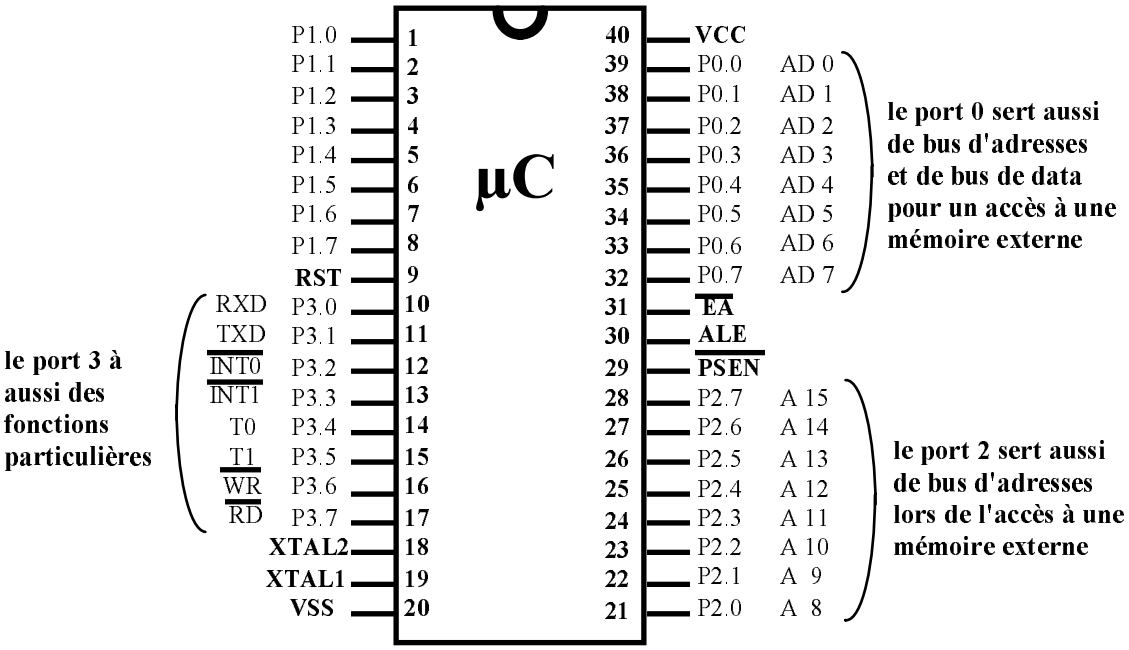


Figure 2 : *Brochage du µC.*

VCC : borne + 5 volts de l'alimentation/

VSS : borne 0 volt de l'alimentation.

XTAL1/2 : broches de câblage du quartz ou d'entrée d'horloge (3,5 à 12 MHz).

P0.0 à P0.7 : bit 0 du port0 au bit 7 du port0.

P1.0 à P1.7 : bit 0 du port1 au bit 7 du port1.

P2.0 à P2.7 : bit 0 du port2 au bit 7 du port2.

P3.0 à P3.7 : bit 0 du port3 au bit 7 du port3.

RST : entrée, patte de reset ("1" de logique actif).

$\overline{EA}$: entrée, (external access) forçage sur mémoire programme externe (0 actif).

ALE : sortie, (address latch enable) de commande de mémorisation des adresses de poids faibles pour accès à une mémoire externe.

$\overline{PSEN}$: sortie, (program store enable) ordre de lecture d'une mémoire programme externe (READ).

Nous remarquerons que les ports 0 et 2 peuvent servir aussi de bus externe au système.

De plus sur le port 3, des notations supplémentaires indiquent des fonctions particulières :

RXD : entrée de la liaison série.

TXD : sortie de la liaison série.

$\overline{INT0}$: entrée de demande d'interruption externe n° 0.

$\overline{INT1}$: entrée de demande d'interruption externe n° 1.

T0 : entrée de comptage n° 0.

T1 : entrée de comptage n° 1.

$\overline{RD}$: sortie ; (read) ordre le lecture d'une mémoire de données externes.

$\overline{WR}$: sortie ; (write) ordre d'écriture sur une mémoire de données externes.

II. Espace mémoire

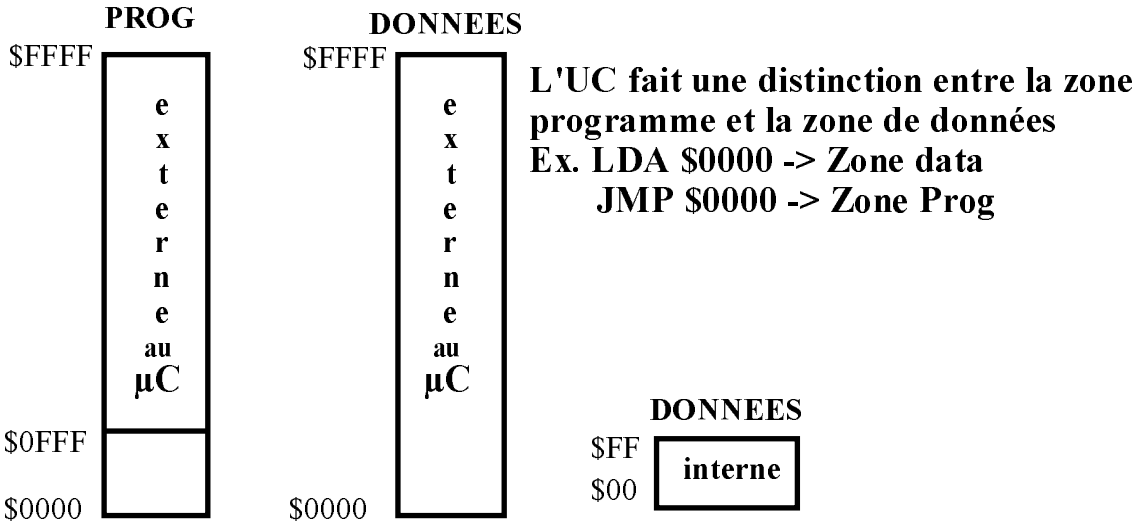


Figure 3 : *Espace mémoire total du µC.*

Ces µC font la distinction entre une zone programme et une zone de stockage de données grâce à l'instruction utilisée. Il sera donc possible d'adresser :

64k de mémoire programme (appelée aussi zone code byte),

+ 64k de mémoire de data externe,

+ 256 octets de data interne.

La zone interne de données (data) est elle même sous divisée en deux zones :

une zone de données de 128 octets,

une zone registre comportant les registres internes de l'UC (accumulateur, stack pointer…) et les registres de gestion des entrées/sorties et des auxiliaires.

II.1. Mémoire programme

64 koctets possibles (avec câblage externe au µC).

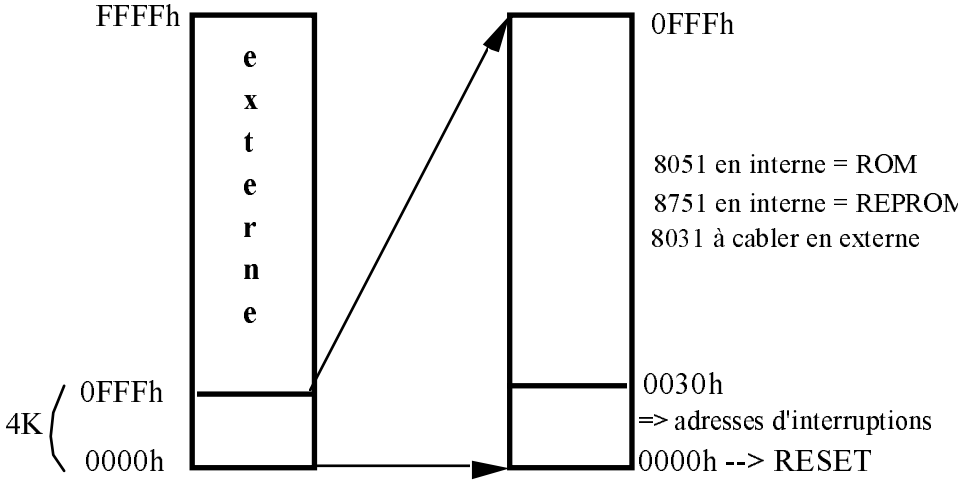


Figure 4 : *Espace mémoire programme.*

L'accès à la mémoire externe est possible grâce aux ports 0 et 2 et signaux de contrôle ALE, $\overline{PSEN}$ et $\overline{EA}$.

II.2.Mémoire de données

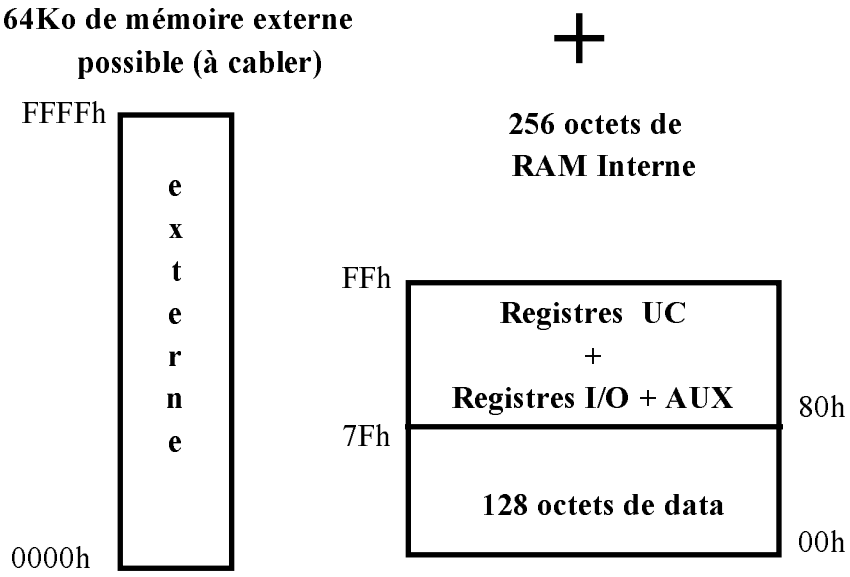


Figure 5 : *Espace mémoire de données.*

Comme pour la mémoire programme, l'accès à la mémoire de données externe est possible grâce aux ports 0 et 2 et signaux de contrôle ALE, $\overline{RD}$ et $\overline{WR}$.

Nous allons étudier en détails l'accès à la RAM interne. Elle se divise en deux parties :

- une zone de 128 octets de données (00h à 7Fh),
- une zone de registres internes et d'accès aux E/S.

II.2.1.Zone de données internes (00h à 7Fh)

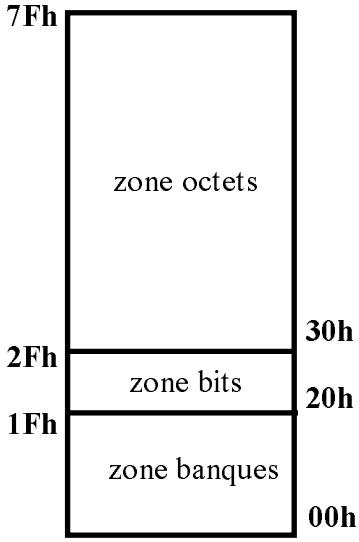
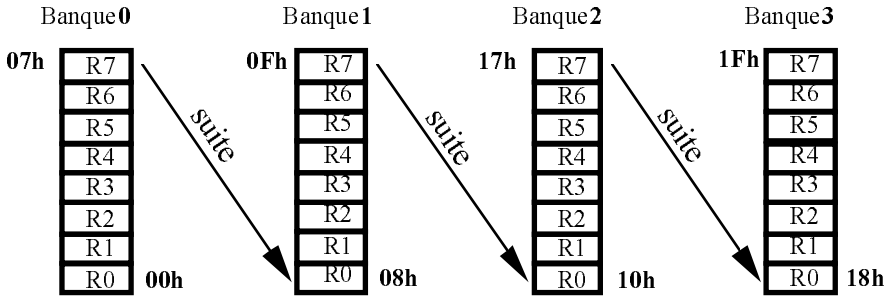


Figure 6 : *Zones de données internes.*

On distingue 3 zones : banques (00h à 1Fh), données à accès bit et octet (20h à 2Fh) et données à accès octet (30h à 7Fh).

Zone banques (00h à 1Fh)

4 banques de travail contenant chacune 8 registres notés de R0 à R7



1°) il faut sélectionner la banque
2°) ensuite on utilise chacun des registres en utilisant son nom
EX: **INC R0** incrémentation de R0
DEC R5 décrémentation de R5

La sélection de la banque se fait par 2 bits **RS0** *et* **RS1**
EX: sélection de la banque 2

```
CLR  RS0 ; clear bit RS0
SETB RS1 ; set bit   RS1
```

Figure 7 : *Zone banques.*

Il est toujours possible d'accéder à un registre de banque par son adresse :

inc 09h Incrémentation de R1 de la banque 1.
(Si la banques est déjà sélectionnées, l'instruction avec le registre de banque est plus rapide que celle avec l'adresse).

Zone de données à accès bit et octet (20h à 2Fh)

7Fh	7Eh	-	-	-	-	-	78h	**2Fh**
-	-	-	-	-	-	-	-	**2Eh**
-	-	-	-	-	-	-	-	**2Dh**
-	-	-	-	-	-	-	-	**2Ch**
-	-	-	-	-	-	-	-	**2Bh**
-	-	-	-	-	-	-	-	**2Ah**
-	-	-	-	-	-	-	-	**29h**
-	-	-	-	-	-	-	-	**28h**
-	-	-	-	-	-	-	-	**27h**
-	-	-	-	-	-	-	-	**26h**
-	-	-	-	-	-	-	-	**25h**
-	-	-	-	-	-	-	-	**24h**
-	-	-	-	-	-	-	-	**23h**
-	-	-	-	-	-	-	10h	**22h**
0Fh	0Eh	0Dh	0Ch	0Bh	0Ah	09h	08h	**21h**
07h	06h	05h	04h	03h	02h	01h	00h	**20h**

Table 1 : *Zone à accès bit et octet.*

Chaque octet de cette zone possède des bits et chacun de ces bits possède un "nom" ou plus exactement une adresse propre permettant au programmeur d'avoir à sa disposition des éléments binaires pour des calculs booléens. Bien entendu les instructions à utiliser

sont des instructions booléennes : ANL (ET logique), ORL (OU logique), CLR (clear bit), SETB (set bit), etc…

Ceci implique qu'il sera possible d'utiliser cette zone par des instructions de type **booléen** portant sur des adresses comprises entre 00h et 7Fh; ou alors des instructions de type **octet** portant sur des adresses comprises entre 20h et 2Fh (manipulant ainsi 8 bits à la fois).

Zone de données à accès octet (30h à 7Fh)

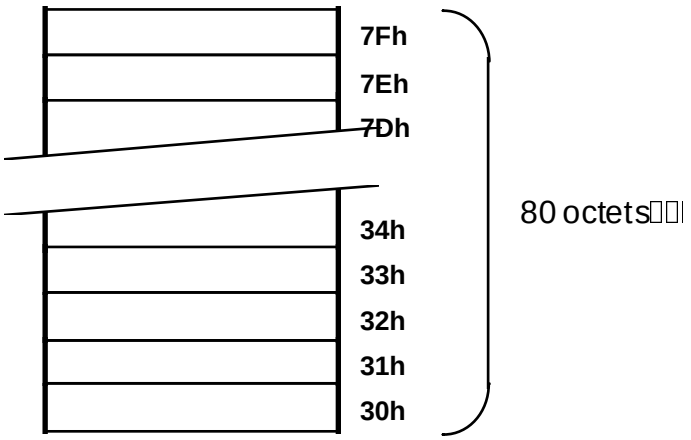


Figure 8 : *Zone à accès octet.*

Il s'agit là d'un accès classique en mémoire de type octet.

II.2.2.Zone de registres internes et d'accès aux E/S

*	**B**	**B.7**	**B.6**	**B.5**	**B.4**	**B.3**	**B.2**	**B.1**	**B.0**	F0h
*	**A**	**ACC. 7**	**ACC. 6**	**ACC. 5**	**ACC. 4**	**ACC. 3**	**ACC. 2**	**ACC. 1**	**ACC. 0**	E0h
*	PSW	**C**	**AC**	**F0**	**RS1**	**RS0**	**OV**	**-**	**P**	D0h
*	**IP**	**-**	**-**	**-**	**PS**	**PT1**	**PX1**	**PT0**	**PX0**	B8h
*	**P3**	**P3.7**	**P3.6**	**P3.5**	**P3.4**	**P3.3**	**P3.2**	**P3.1**	**P3.0**	B0h
*	**IE**	**EA**	**-**	**-**	**ES**	**ET1**	**EX1**	**ET0**	**EX0**	A8h
*	**P2**	**P2.7**	**P2.6**	**P2.5**	**P2.4**	**P2.3**	**P2.2**	**P2.1**	**P2.0**	A0h
	SBUF									99h
*	**SCON**	**SM0**	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**	98h
*	**P1**	**P1.7**	**P1.6**	**P1.5**	**P1.4**	**P1.3**	**P1.2**	**P1.1**	**P1.0**	90h
	TH1									8Dh
	TH0									8Ch
	TL1									8Bh
	TL0									8Ah
	TMOD	*GATE*	*C/$\overline{T}$*	*M1*	*M0*	*GATE*	*C/$\overline{T}$*	*M1*	*M0*	89h
*	**TCON**	**TF1**	**TR1**	**TF0**	**TR0**	**IE1**	**IT1**	**IE0**	**IT0**	88h
	PCON	*SMOD*	-	-	-	*GF1*	*GF0*	*PD*	*IDL*	87h
	DPH									83h
	DPL									82h
	SP									81h
*	**P0**	**P0.7**	**P0.6**	**P0.5**	**P0.4**	**P0.3**	**P0.2**	**P0.1**	**P0.0**	80h

Table 2 : *Zone de registres internes et d'entrées-sorties.*

Chacun de ces registres est accessible par son nom ou son adresse. Ceux qui ont une adresse divisible par 8 (X0h ou X8h) sont aussi accessibles bit.

II.3. Exemple : l'adresse 0

Mémoire programme : movc A,@A+DPTR
Interne si $\overline{EA}$=1, externe si $\overline{EA}$=0.
=> 2 adresses possibles.

Mémoire de données externe : movx A,@A+DPTR
=> 1 adresse possible.

Mémoire de données interne : mov 00h,#32h transfert d'octet
mov C,00h transfert de bit (bit C intermédiaire)
=> 2 adresses possibles.

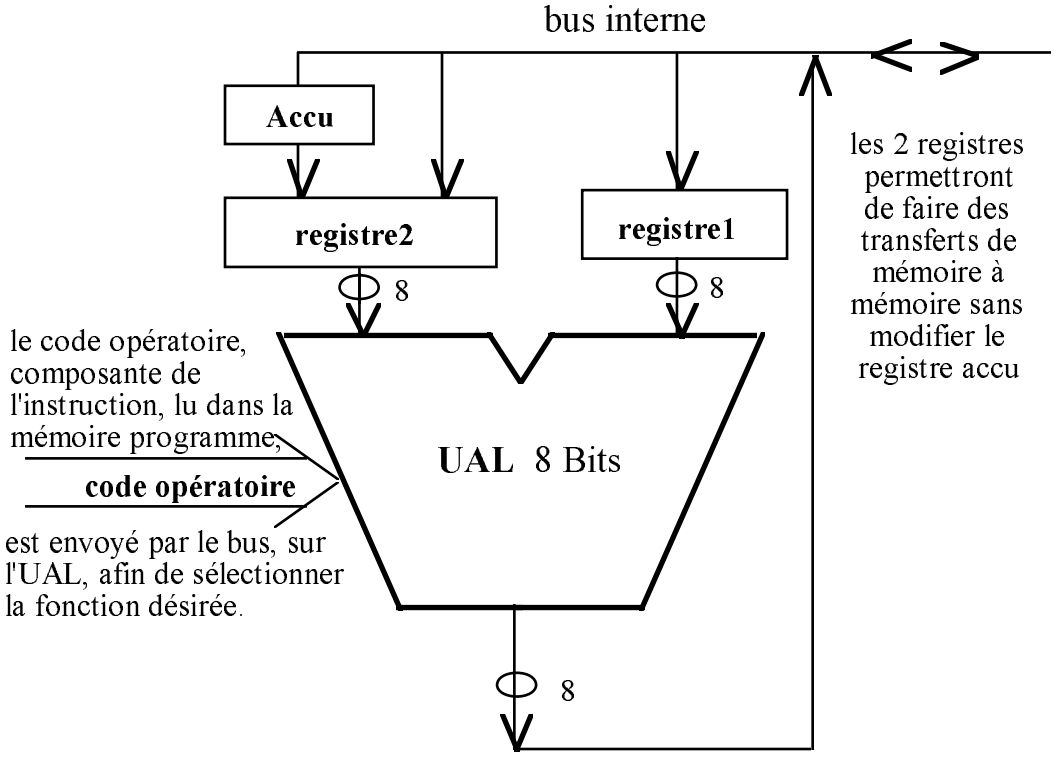
=> L'adresse 0 peut correspondre à 5 "objets" différents. C'est le type d'instruction qui permet de faire la distinction.

III. Le microprocesseur

III.1. Constitution

III.1.1. Unité Arithmétique et Logique (UAL)

Elle reçoit deux opérandes plus le code opératoire (instruction) correspondant à la fonction logique ou arithmétique à exécuter sur ces deux opérandes. Elle fournit le résultat de l'opération demandée qui, généralement, sera stocké temporairement dans un registre lui appartenant.



cette UAL permettra d'exécuter des instructions de multiplication ou de division

Figure 9 : *Unité Arithmétique et Logique.*

Le registre, ou accumulateur (Accu), A joue un rôle très particulier pour l'UAL : il intervient dans beaucoup d'instruction. Le nombre limité d'instructions (code sur 8 bits pour limiter la taille des programmes) impose des choix.

III.1.2. Accumulateur A

C'est le registre de travail par excellence. Beaucoup d'instructions l'utilisent en tant qu'opérande et en tant que registre temporaire de réception du résultat. Il est accessible bit.

E7h	E6h	E5h	E4h	E3h	E2h	E1h	E0h	adresse
ACC.7	**ACC.6**	**ACC.5**	**ACC.4**	**ACC.3**	**ACC.2**	**ACC.1**	**ACC.0**	E0h

Table 3 : *Registre A.*

III.1.3. Accumulateur B

Il est utilisé spécifiquement pour deux instructions : multiplication et division. Il peut servir aussi de registre de stockage provisoire. Il est accessible bit.

F7h	F6h	F5h	F4h	F3h	F2h	F1h	F0h	adresse
B.7	**B.6**	**B.5**	**B.4**	**B.3**	**B.2**	**B.1**	**B.0**	F0h

Table 4 : *Registre B.*

III.1.4. Registre d'état (PSW : Program Status Word)

Il renseigne sur le déroulement des instructions. Il contient des indicateurs (appelés aussi flags ou drapeaux). Il est accessible bit.

D7h	D6h	D5h	D4h	D3h	D2h	D1h	D0h	adresse
C	**AC**	**F0**	**RS1**	**RS0**	**OV**	**-**	**P**	D0h

Table 5 : *Registre d'état (PSW).*

C (D7h) : **CARRY.** Indique s'il est à 1 un débordement sur le format binaire.

AC (D6h) : **CARRY auxiliaire**. Indique s'il est à 1 un débordement sur 4 bits de poids faibles utilisés pour la conversion Décimal Codé Binaire(DCB).

F0 (D5h) : **FLAG USER**. Bit disponible pour l'utilisateur.

RS1 (D4h) et **RS0** (D3h) :

Registers selected. Choix de la banque de travail (une banque correspond à 8 registres de R0 à R8)

RS1	RS0	Banque
0	0	0
0	1	1
1	0	2
1	1	3

Table 6 : *Choix de la banque active.*

OV (D2h) : Overflow. Indique le débordement sur nombres signés.

P (D0h) : **Parité.**

P=1 si l'Accu contient un nombre impair de bits à 1

P=0 si l'Accu contient un nombre pair de bits à 1

EX: A = 02h (00000010) => P = 1

A = 03h (00000011) => P = 0

III.1.5. Pointeur de pile (SP : Stack Pointer)

Registre de gestion de la pile. Le stack pointer gère automatiquement la pile lors de l'appel de sous programmes ou lors de déclenchement de programmes d'interruption afin d'enregistrer l'adresse de retour dans le programme principal.

Des instructions spécifiques à la pile génèrent automatiquement l'incrémentation ou la décrémentation du SP. Lors d'une opération de stockage dans la pile le SP est incrémenté. Lors d'une opération de lecture dans la pile le SP est décrémenté. Ces opérations se déroulent en 2 temps : EX:

instruction PUSH (stockage)	instruction POP (lecture)
1) SP=SP+1	1) lecture dans la pile
2) écriture dans la pile (stockage)	2) SP=SP-1

Le SP contient donc l'adresse de la dernière valeur rentrée. On dit qu'il "pointe" sur la dernière valeur rentrée.

Attention : au Reset le SP est initialisé à 07h. Ceci implique que si l'on utilise la pile lors de PUSH ou de POP, ou bien lors d'appel de sous programmes, on risque d'écraser la zone banque de travail de la RAM interne. Donc généralement, par une instruction de chargement du SP, on modifiera sa valeur (bien souvent avec la valeur 30h, ce qui correspond à une adresse de pile placée au dessus de la zone bits de la RAM), afin de ne pas avoir de conflit.

Remarque : les instructions PUSH et POP travaillent en adressage direct; ceci veut dire qu'il faut leur fournir l'adresse de la case mémoire qu'ils doivent sauvegarder ou restituer. Le symbole A n'est pas une adresse => l'instruction PUSH A n'existe pas, par contre, PUSH 0E0h est correct. L'assembleur connaît aussi le label ACC comme étant la valeur 0E0h, il est donc plus clair d'utiliser PUSH ACC.

III.1.6.Pointeur de données (DPTR : Data PoinTer Register)

Registre 16 bits permettant l'accès à la zone de données extérieur et à la zone de programme (interne ou externe).

III.2.Modes d'adressage

Même idée que 68000. On ne voit que la syntaxe.

Immédiat : mov A, **#5Ch**

Registre : mov **R1**, #47

Direct : mov **56h**, #96h

Indirect : mov **@R0**, #B2h

III.3.Différents types d'instructions

III.3.1.Arithmétiques

ADD	addition de 2 octets (positionne le carry)
ADDC	addition de 2 octets + carry (positionne le carry)
SUBB	soustraction de 2 octets - carry (borrow, retenue) (positionne le carry)
INC	incrémentation (ne positionne pas le carry)
DEC	décrémentation (ne positionne pas le carry)
MUL	multiplication (A x B)=>A contient le poids faible du résultat, B le poids fort
DIV	division (A : B)=>A contient le quotient et B le reste
DA	décimal adjust (ajustement au code DCB de l'Accu)

Cette dernière instruction doit être utilisée après une addition de nombres déjà codés en DCB. Cette addition positionne le drapeau de carry auxiliaire et de carry indiquant un débordement de format permettant au processeur de faire l'ajustement en DCB.

III.3.2.Logiques sur octets

ANL	ANd Logical (et logique)
ORL	OR Logical (ou logique)
XRL	eXclusive oR Logical (ou exclusif)
CLR A	CLeaR A (raz de l'accu)
CPL A	ComPLement A (complément à 1, inversion des bits de l'octet)
RL A	Rotate A on the Left (rotation de 1 bit vers la gauche de l'octet)
RLC A	Rotate A on the Left with Carry (rotation sur 9 bits de 1 bit vers la gauche, le carry fournit le 9ème bit)
RR A	Rotate A on the Right (rotation de 1 bit vers la droite de l'octet)
RRC A	Rotate A on the Right with Carry (rotation sur 9 bits de 1 bit vers la droite, le carry fournit le 9ème bit)

SWAP A échange des 4 bits de poids forts avec les 4 bits de poids faibles de A.

III.3.3. Transfert sur octets

MOV MOVe (copie une valeur en utilisant la mémoire RAM interne)
MOVX MOVe eXternal (copie une valeur en utilisant la mémoire data externe)
MOVC MOVe Code byte (copie une valeur en utilisant la mémoire programme)
PUSH pousse dans la pile le contenu d'une adresse (adresse de A = E0h ou label ACC).
POP retire le contenu de la pile et le met à l'adresse spécifiée
XCH eXCHange (échange l'Accu avec 1 registre ou une case mémoire)
XCHD eXCHange Digit (échange les 4 bits de poids faibles)

III.3.4. Booléennes

CLR CLeaR (raz du bit)
SETB SET Bit (mise a 1 du bit)
CPL ComPLement (complément du bit)
ANL ANd Logical (et logique entre le carry et un bit ou un $\overline{bit}$)
ORL OR Logical (ou logique entre le carry et un bit ou un $\overline{bit}$)
MOV transfert de bit (soit carry vers bit, soit bit vers carry)

III.3.5. Modification du PC

III.3.5.1. sauts inconditionnels

Le compteur programme est chargé avec une nouvelle valeur sans condition.

De type JUMP : BRA en 68000
- LJMP Long JuMP (saut absolu à une adresse spécifiée sur 16 bits)
- AJMP Absolute JuMP (saut absolu à une adresse spécifiée sur 11 bits)
- SJMP Short JuMP (saut court et relatif)
- JMP @A+DPTR JuMP indirect

De type CALL : BSR en 68000
- LCALL Long CALL (saut absolu à une adresse spécifiée sur 16 bits)
- ACALL Absolu CALL (saut absolu à une adresse spécifiée sur 11 bits)

De type retour :
- RET RETurn (return from sub routine retour de sous programme)
- RETI RETurn from Interrupt

III.3.5.2. sauts conditionnels

Sur octets :
- JZ Jump if Zero (saut si l'accu = 0)
- JNZ Jump if Not Zéro (saut si l'accu ≠ 0)
- CJNE Compare and Jump if Not Equal (compare et saute si ≠)
- DJNZ Decrement and Jump if Not Zero (décrémente et saute si ≠ 0)

Sur bits :
- JC Jump if Carry (saut si le carry = 1)
- JNC Jump if Not Carry (saut si le carry = 0)
- JB Jump if Bit (saut si le bit dont l'adresse ou le nom est spécifié = 1)
- JNB Jump if Not Bit (saut si le bit dont l'adresse ou le nom est spécifié = 0)
- JBC Jump if Bit and Clear bit (saut si le bit dont l'adresse ou le nom est spécifié = 1 et raz du bit)

III.3.6. Spéciale

NOP Not OPeration (ne rien faire).

III.3.7. Instructions qui modifient les drapeaux du PSW

instruction	drapeaux			instruction	drapeaux		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

Table 7 : *Instructions modifiant les drapeaux.*

Il n'existe pas d'instruction de comparaison directe. Une solution revient à utiliser l'instruction CJNE qui non seulement génère un saut mais positionne le bit C :

CJNE A, #Donnée, Etiquette

Exécute un branchement à Etiquette si A≠Donnée
et positionne le bit C : C=1 si A<Donnée
C=0 si A≥Donnée

```
Ex :        CJNE    A, #34h, Suite
      Suite:JC      Ainféieurà34    ; Saute si C=1 à Ainférieurà34.
                                    ; Sinon, continue.
```

III.4. Liste des instructions

instructions arithmétiques		description	bytes/	cycles	code opératoire (hex)
ADD	A,Rr	A + registre --> A	1	1	2*
ADD	A,direct	A + octet direct --> A	2	1	25
ADD	A,@Ri	A + octet indirect en RAM --> A	1	1	26,27
ADD	A,#data	A + donnée immédiate --> A	2	1	24
ADDC	A,Rr	A + registre + carry --> A	1	1	3*
ADDC	A,direct	A + octet direct + carry --> A	2	1	35
ADDC	A,@Ri	A + octet indirect en RAM + carry --> A	1	1	36,37
ADDC	A,#data	A + donnée immédiate + carry--> A	2	1	34
SUBB	A,Rr	A - registre - carry --> A	1	1	9*
SUBB	A,direct	A - octet direct - carry --> A	2	1	95
SUBB	A,@Ri	A - octet indirect en RAM - carry --> A	1	1	96,97
SUBB	A,#data	A - donnée immédiate - carry--> A	2	1	94
INC	A	incrémentation de A	1	1	04
INC	Rr	incrémentation du registre	1	1	0*
INC	direct	incrémentation de l'octet direct	2	1	05
INC	@Ri	incrémentation de l'octet indirect en RAM	1	1	06,07
DEC	A	décrémentation de A	1	1	14

DEC	Rr	décrémentation du registre	1	1	1*
DEC	direct	décrémentation de l'octet direct	2	1	15
DEC	@Ri	décrémentation de l'octet indirect en RAM	1	1	16,17
INC	DPTR	incrémentation de DPTR	1	2	A3
MUL	AB	multiplication de A par B	1	4	A4
DIV	AB	division de A par B	1	4	84
DA	A	ajustement décimal de A	1	1	D4

instructions logiques	**description**	**bytes/ cycles**		**code opératoire (hex)**
ANL A,Rr	A ✋ (registre --> A	1	1	5*
ANL A,direct	A ✋ octet direct --> A	2	1	55
ANL A,@Ri	A ✋ octet indirect en RAM --> A	1	1	56,57
ANL A,#data	A ✋ donnée immédiate --> A	2	1	54
ANL direct,A	octet direct ✋ A --> octet direct	2	1	52
ANL direct,#data	octet direct ✋ donnée immédiate --> octet direct	3	2	53
ORL A,Rr	A ✞ registre --> A	1	1	4*
ORL A,direct	A ✞ octet direct --> A	2	1	45
ORL A,@Ri	A ✞ octet indirect en RAM --> A	1	1	46,47
ORL A,#data	A ✞ donnée immédiate --> A	2	1	44
ORL direct,A	octet direct ✞ A --> octet direct	2	1	42
ORL direct,#data	octet direct ✞ donnée immédiate --> octet direct	3	2	43
XRL A,Rr	A ⊕ registre --> A	1	1	6*
XRL A,direct	A ⊕ octet direct --> A	2	1	65
XRL A,@Ri	A ⊕ octet indirect en RAM --> A	1	1	66,67
XRL A,#data	A ⊕ donnée immédiate --> A	2	1	64
XRL direct,A	octet direct ⊕ A --> donnée	2	1	62
XRL direct,#data	octet direct ⊕ donnée immédiate --> octet direct	3	2	63
CLR A	RAZ de A	1	1	E4
CPL A	complément de A	1	1	F4
RL A	rotation de A sur la gauche	1	1	23
RLC A	rotation de A avec le carry sur la gauche	1	1	33
RR A	rotation de A sur la droite	1	1	03
RRC A	rotation de A avec le carry sur la droite	1	1	13
SWAP A	échange bits faibles avec bits forts dans A	1	1	C4

instructions de transfert de données	description	bytes/ cycles	code opératoire (hex)
MOVA,Rr	copie registre --> A	1 1	E*
MOVA,direct	copie octet direct --> A	2 1	E5
MOVA,@Ri	copie octet indirect en RAM --> A	1 1	E6,E7
MOVA,#data	copie donnée immédiate --> A	2 1	74
MOVRr,A	copie A --> registre	1 1	F*
MOVRr,direct	copie octet direct --> Rr	2 2	A*
MOVRr,#data	copie donnée immédiate --> Rr	2 1	7*
MOVdirect,A	copie A --> octet direct	2 1	F5
MOVdirect,Rr	copie Rr --> octet direct	2 2	8*
MOVdirect1,direct2	copie octet direct2 --> octet direct1	3 2	85
MOVdirect,@Ri	copie octet indirect en RAM --> octet direct	2 2	86,87
MOVdirect,#data	copie donnée immédiate --> octet direct	3 2	75
MOV@Ri,A	copie A --> octet indirect en RAM	1 1	F6,F7
MOV@Ri,direct	copie octet direct --> octet indirect en RAM	2 2	A6,A7
MOV@Ri,#data	copie donnée imm. --> octet indirect en RAM	2 1	76,77
MOVDPTR,#data16	copie donnée immédiate / 16 bits --> DPTR	3 2	90
MOVC A,@A+DPTR	copie l'octet prog. indirect relatif à DPTR --> A	1 2	93
MOVC A,@A+PC	copie l'octet prog. indirect relatif au PC --> A	1 2	83
MOVX A,@Ri	copie octet indirect externe selon Ri --> A	1 2	E2,E3
MOVX A,@DPTR	copie octet indirect externe selon DPTR --> A	1 2	E0
MOVX @Ri,A	copie A --> octet indirect externe selon Ri	1 2	F2,F3
MOVX @DPTR,A	copie A --> octet indirect externe selon DPTR	1 2	F0
PUSH direct	copie octet direct --> pile	2 2	C0
POP direct	copie la pile --> octet direct	2 2	D0
XCH A,Rr	échange le registre avec A	1 1	C*
XCH A,direct	échange l'octet direct avec A	2 1	C5
XCH A,@Ri	échange l'octet indirect en RAM avec A	1 1	C6,C7
XCHD A,@Ri	échange bits de poids faibles de l'oct. ind. avec A	1 1	D6,D7

instructions booléennes		description	bytes/ cycles		code opératoire (hex)
CLR	C	RAZ du carry	1	1	C3
CLR	bit	RAZ du bit	2	1	C2
SETB	C	mise à 1 du carry	1	1	D3
SETB	bit	mise à 1 du bit	2	1	D2
CPL	C	complémente le carry	1	1	B3
CPL	bit	complémente le bit	2	1	B2
ANL	C,bit	Carry ✋ (bit --> Carry	2	2	82
ANL	C,/bit	Carry ✋ complément du bit --> Carry	2	2	B0
ORL	C,bit	Carry ✞ bit --> Carry	2	2	72
ORL	C,/bit	Carry ✞ complément du bit --> Carry	2	2	A0
MOV	C,bit	copie le bit --> Carry	2	1	A2
MOV	bit,C	copie le Carry --> bit	2	2	92

instructions de saut		description	bytes/ cycles		code opératoire (hex)
ACALL	adr11	appel à sous-programme avec adr 11 bits	2	2	∞1adr
LCALL	adr16	appel à sous-programme avec adr 16 bits	3	2	12
RET		retour de sous-programme	1	2	22
RETI		retour d'interruption	1	2	32
AJMP	adr11	saut avec adr 11 bits	2	2	•1adr
LJMP	adr16	saut avec adr 16 bits	3	2	02
SJMP	rel	saut relatif (+127,-128)	2	2	80
JMP	@A+DPTR	saut indirect à l'adr donnée par A+DPTR	1	2	73
JZ	rel	saut si A = 0	2	2	60
JNZ	rel	saut si A ≠ 0	2	2	70
JC	rel	saut si carry = 1	2	2	40
JNC	rel	saut si carry = 0	2	2	50
JB	bit,rel	saut si le bit = 1	3	2	20
JNB	bit,rel	saut si le bit = 0	3	2	30
JBC	bit,rel	saut si le bit = 1 et RAZ du bit	3	2	10
CJNE	A,direct,rel	saut si A ≠ octet direct	3	2	B5
CJNE	A,#data,rel	saut si A ≠ de la valeur immédiate	3	2	B4
CJNE	Rr,#data,rel	saut si le registre ≠ de la valeur immédiate	3	2	B*
CJNE	@Ri,#data,rel	saut si l'octet indirect ≠ de la valeur immédiate	3	2	B6,B7
DJNZ	Rr,rel	décrémente et saut si le registre ≠ 0	2	2	D*
DJNZ	direct,rel	décrémente et saut si l'octet direct ≠ 0	3	2	D5
NOP		pas d'opération	1	1	00

Codes hexadécimaux correspondants aux symboles utilisés dans la liste des instructions :

* : 8,9,A,B,C,D,E,F.
• : 01,21,41,61,81,A1,C1,E1.
∞ : 11,31,51,71,91,B1,D1,F1.

Modes d'adressage :

Rr :	registre de travail de R0 à R7
direct :	soit un des octets de la mémoire RAM interne ou un registre de fonction .
@Ri :	adressage indirect de la RAM interne par registre R0 ou R1.
#data :	constante de 8 bits contenue dans l'instruction.
#data16 :	constante de 16 bits contenue dans l'instruction.
bit :	bit adressé directement et contenu dans la mémoire RAM ou un registre de fonction.
adr16 :	adresse de destination sur 16 bits (branchement dans les 64k de la mémoire programme).
adr11†:	adresse de destination sur 11 bits (branchement dans une page de 2 koctets).
rel :	adressage relatif de -128 octets en arrière ou +127 octets en avant.

IV. Les entrées-Sorties

IV.1. Constitution

IV.1.1. Liaisons vers l'extérieur

4 ports P0 à P3 de 8 bits chacun
(ceci correspond aux pattes du circuit)

1 liaison série
TXD (sortie : transmit data)
RXD (entrée : receive data)

2 demandes externes d'interruption
$\overline{\text{INT0}}$ (entrée : interrupt 0)
$\overline{\text{INT1}}$ (entrée : interrupt 1)

2 entrées de comptage
T0 (entrée : timer 0)
T1 (entrée : timer 1)
Ceci implique que nous aurons 2 compteurs afin de comptabiliser le nombre d'impulsions qui seront reçues sur ces entrées.

IV.1.2. Registres internes

4 registres appelés P0 à P3
C'est leur contenu qui sera copié sur les pattes correspondantes du µC lors d'une écriture sur le port.

Systèmes de comptage : 2 timers (compteurs) de 16 bits chacun et de registres de contrôle:
- TIMER 0 décomposé en 2 sous timers : TL0 (timer n°0 poids faible)
  TH0 (timer n°0 poids fort)
- TIMER 1 décomposé en 2 sous timers : TL1 (timer n°1 poids faible)
  TH1 (timer n°1 poids fort)
- TMOD (timer mode) : Registre de choix du mode de fonctionnement.
- TCON (timer control) : Registre de contrôle du fonctionnement.

Une liaison série
- SBUFF (serial buffer) : Registre dans lequel nous écrirons pour transmettre une information ou que nous lirons lors d'une réception.
- SCON (serial control) : Registre de gestion de la liaison série.

Des registres de gestion des interruptions
- IE (interrupt enable) : Registre de validation)

- IP (interrupt priority) : Registre de choix de priorité)

Un registre de contrôle de l'énergie consommée

- PCON (power control) : Contrôle de l'énergie (réduction de consommation).

IV.2.Ports d'entrées-sorties

Par port il faut distinguer les pattes du circuit des registres associés dont la valeur est recopiée sur les pattes du circuit.

P0 (80h) avec 8 bits adressables notés P0.0 à P0.7 (80h à 87h)

P1 (90h) avec 8 bits adressables notés P1.0 à P1.7 (90h à 97h)

P2 (A0h) avec 8 bits adressables notés P2.0 à P2.7 (A0h à A7h)

P3 (B0h) avec 8 bits adressables notés P3.0 à P3.7 (B0h à B7h)

Particularité du port 3 : Chaque bit de ce port, outre sa fonction port, possède en plus une fonction supplémentaire.

B7h	B6h	B5h	B4h	B3h	B2h	B1h	B0h	
$\overline{\text{RD}}$	$\overline{\text{WR}}$	T1	T0	$\overline{\text{INT1}}$	$\overline{\text{INT0}}$	TXD	RXD	B0h

Table 8 : *Port 3.*

RXD : Réception sur la liaison série

TXD : Transmission sur la liaison série

$\overline{\text{INT0}}$: Demande externe d'interruption n°0

$\overline{\text{INT1}}$: Demande externe d'interruption n°1

T0 : Connexion possible sur Timer0, comptage d'impulsions externes.

T1 : Connexion possible sur Timer1, comptage d'impulsions externes.

$\overline{\text{WR}}$: Ordre d'écriture sur mémoire de data externe.

$\overline{\text{RD}}$: Ordre de lecture sur mémoire de data externe.

IV.3.Système de comptage

IV.3.1.Constitution

2 timers 16 bits (compteurs) chacun étant décomposés en 2 sous-ensembles de 8 bits :

	8Ch	8Ah		8Dh	8Bh
TIMER 0	TH0	TL0	TIMER 1	TH1	TL1

Table 9 : *Deux compteurs 16 bits.*

2 registres de configuration et de contrôle :

TMOD Registre de choix du mode de fonctionnement des TIMERS.

TCON Registre de contrôle des TIMERS et des demandes d'interruption externes.

IV.3.2.Fonctionnement

Choix de l'horloge : Par le bit $C/\overline{T}$ de TMOD

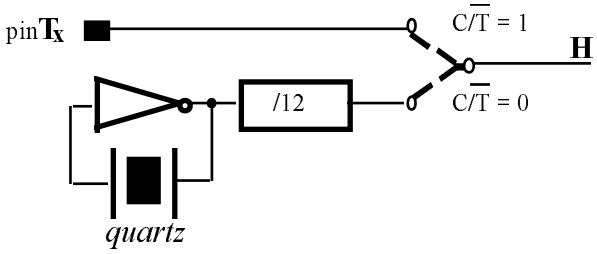


Figure 10 : *Choix de l'horloge des compteurs.*

$C/\overline{T} = 1$ => La patte T_X fournit l'horloge.

$C/\overline{T} = 0$ => La fréquence du quartz/12 correspond à l'horloge.

Contrôle du fonctionnement : C'est une fonction logique qui contrôle l'interrupteur

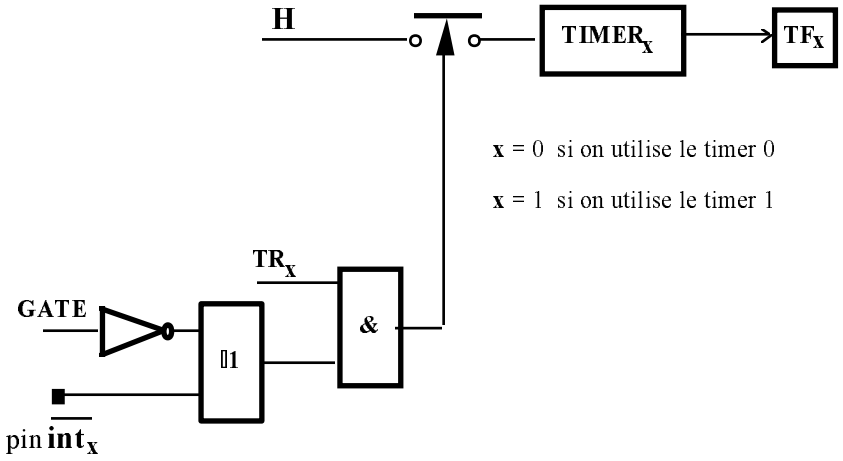


Figure 11 : *Contrôle du fonctionnement des compteurs.*

GATE : Bit de TMOD (autorisation du contrôle de fonctionnement par l'extérieur).

pin $\overline{\textbf{INT}_X}$: Patte du µC (la demande d'interruption externe peut, aussi contrôler le fonctionnement du TIMER).

TR$_X$: TIMER RUN bit de TCON permettant un contrôle logiciel du TIMER (setb TRx)

Conditions de fonctionnement

GATE=0 (on ferme la porte).

Il suffit de faire TR_X =1 (timer run) pour que le timer puisse compter

GATE=1 (on ouvre la porte)

Il faut que TR_X =1 et que la patte $\overline{INT_X}$ du circuit =1

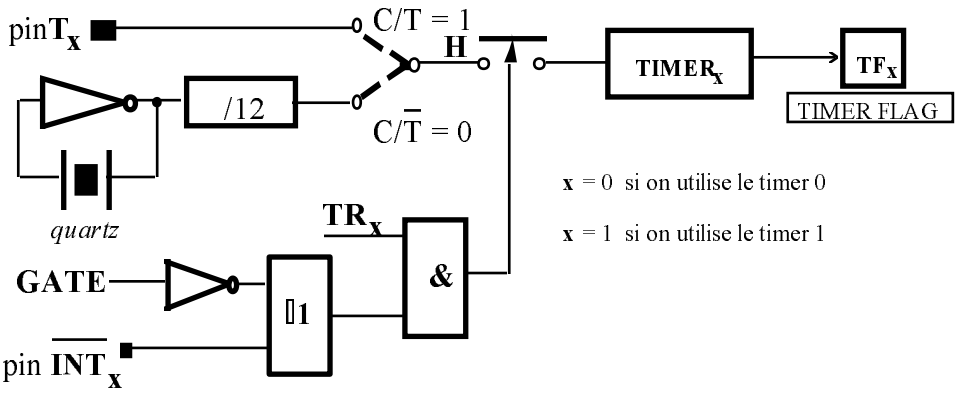


Figure 12 : *Conditions de fonctionnement des compteurs.*

Lorsque le TIMER passe par sa plus grande valeur et retombe à zéro le TIMER FLAG (indicateur de fin de comptage noté TF_X) se positionne à 1

Choix et étude du mode de fonctionnement

4 modes possibles. Le choix se fait par l'intermédiaire de 2 bits, M1 et M0 situés dans TMOD

M1	M0	mode
0	0	0
0	1	1

1	0	2
1	1	3

Table 10 : *Choix du mode de fonctionnement des compteurs.*

Dans tous les cas, le timer flag est positionné par "hard" à 1, la remise à 0 se fait par programme.

Mode 0

Ce mode configure le $TIMER_X$ en un compteur sur 13 bits

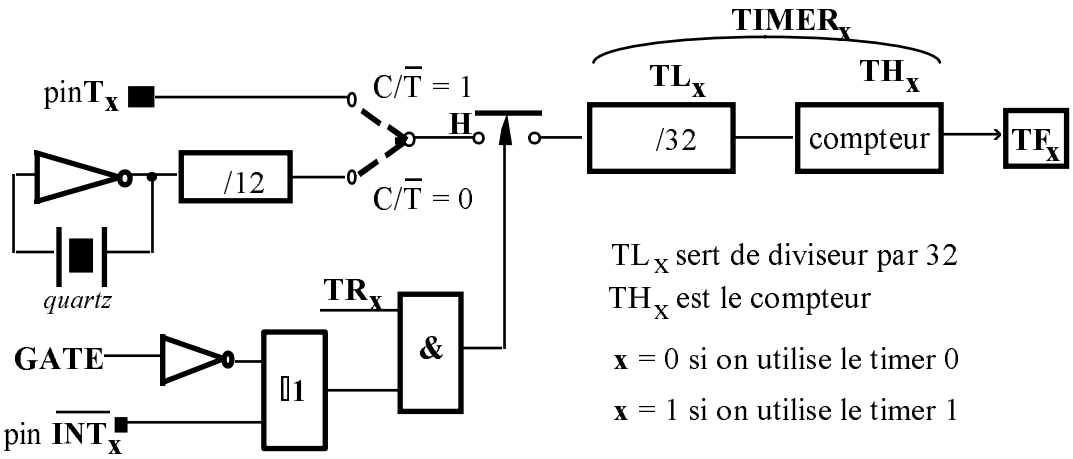


Figure 13 : *Compteur en mode 0.*

Dans ce cas particulier il faut considérer le système comme étant composé d'une horloge/32 suivie d'un compteur sur 8 bits. Même si le diviseur est constitué par la partie faible du timer il n'est pas intéressant de venir gérer cette partie là. Cette configuration sert généralement pour une temporisation qui ne nécessite pas une grande précision. L'avantage est qu'il ne faut gérer qu'un seul octet (THx).

Mode 1

Ce mode configure le TIMERx en un compteur sur 16 bits

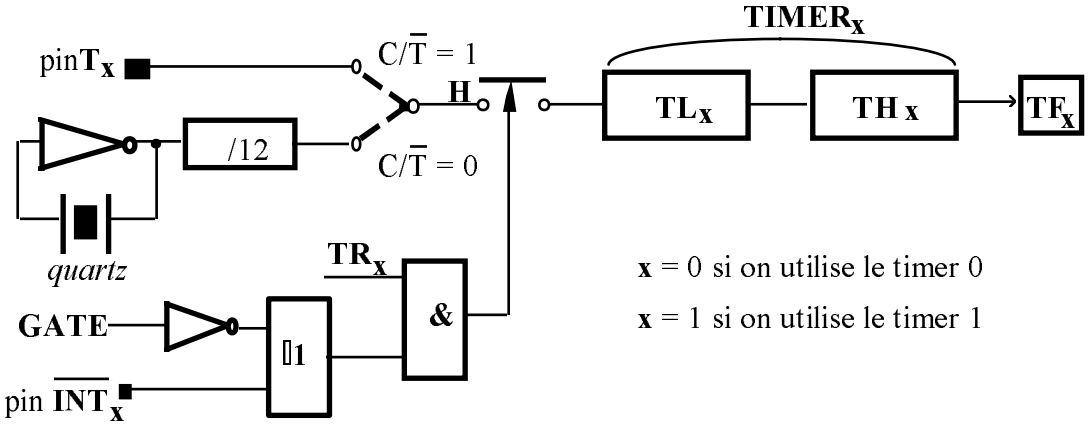


Figure 14 : *Compteur en mode 1.*

Comme pour des compteurs réalisés avec des bascules, TL_X est la partie faible du compteur (la plus proche de l'horloge) et TH_X est la partie poids forts du compteur (la plus éloignée de l'horloge).

Mode 2

Ce mode configure le $TIMER_X$ en un système avec rechargement automatique. Dès que le compteur arrive à zéro. TL_X est le compteur et TH_X est le registre dont la valeur est rechargée dans le compteur.

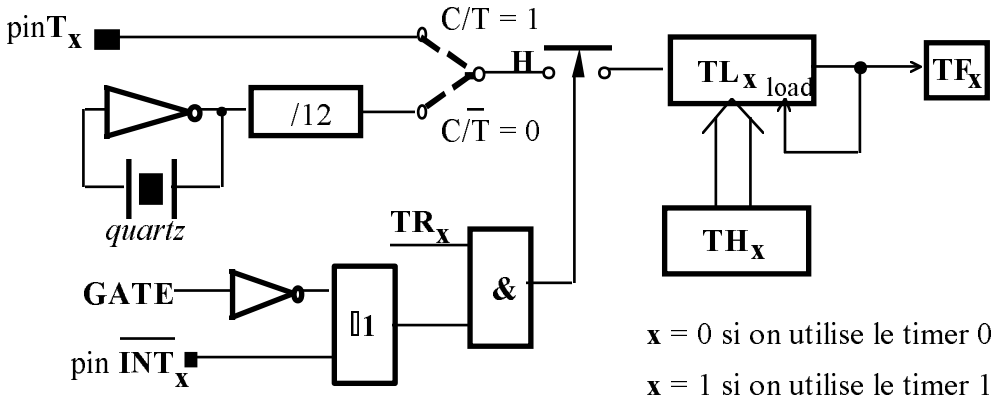


Figure 15 : *Compteur en mode 2.*

Dans ce mode il suffit d'initialiser une seule fois TH_X et chaque fois que TL_X arrive à zéro il est automatiquement rechargé avec la valeur de TH_X.

C'est ce mode là qui est le mode préférentiel de gestion de la vitesse de transmission pour la liaison série et c'est le TIMER1 qui est imposé définir la vitesse de transmission.

Mode 3

Dans ce mode le timer0 se sépare en 2 timers 8 bits.

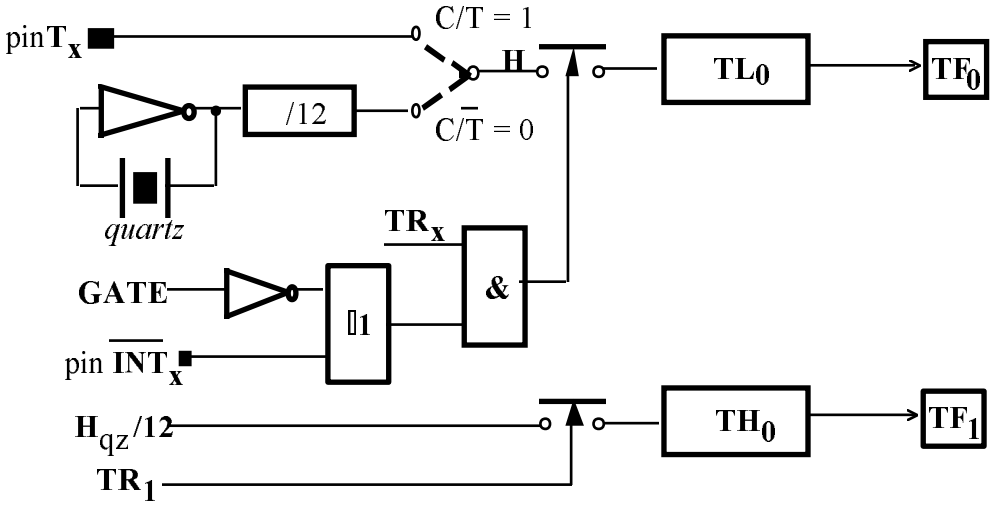


Figure 16 : *Compteur en mode 3.*

Comme nous pouvons le constater le timer0 a emprunté au timer1 le bit timer flag et le bit timer run. Ceci implique que lorsque le timer0 est positionné en mode 3 nous ne pouvons plus nous rendre compte si le timer1 a fini de compter et nous ne pouvons plus contrôler le marche/arrêt du timer1 par le bit timer run (TR).

De ce dernier fait, si le timer0 est en mode 3 et si le timer1 est aussi en mode 3, le timer1 est arrêté.

Si le timer0 est en mode 3 et si le timer1 est dans un mode ≠ 3, le timer1 fonctionne dans le mode sélectionné mais il ne possède plus de TF1 et pour arrêter le timer1, il faut le positionner en mode 3.

Registres Associés aux timers

TMOD : Registre de choix du mode de fonctionnement
Accessible par l'adresse 89h (mais pas accessible bit)

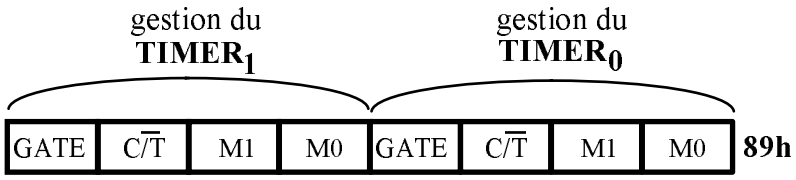


Figure 17 : *Registre TMOD.*

GATE : Validation du contrôle externe de marche/arrêt du µC

C/$\overline{T}$: Choix horloge du quartz ou externe

M1 et M0 : choix du mode de fonctionnement

TCON†: Registre de contrôle de fonctionnement des timers et de contrôle des demandes d'interruptions externes.

Accessible par l'adresse 88h (adressable bit 88h à 8Fh)

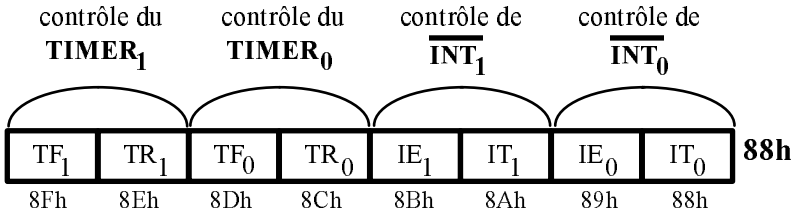


Figure 18 : *Registre TCON.*

TF_X : Indicateur (flag) de passage à zéro du $TIMER_X$.

TR_X : Contrôle du marche/arrêt du $TIMER_X$.

IE_X : Indicateur (flag) de demande d'interruption externe INT_X.

IT_X : Choix de l'événement externe qui crée la demande d'interruption.

Si ITx = 1 => demande d'interruption externe sur front descendant.

Si ITx = 0 => demande d'interruption externe sur niveau bas.

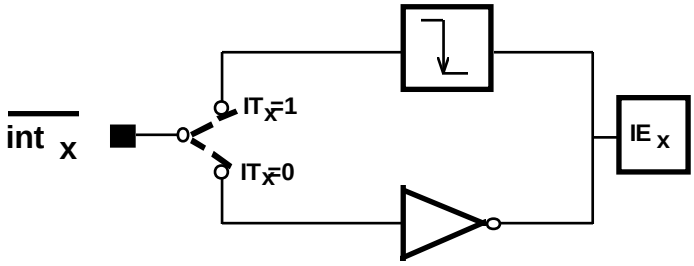


Figure 19 : *Choix de l'événement générateur d'interruption pour les compteurs.*

Exemple d'utilisation d'un timer

Problème : Avoir le positionnement du timer flag au bout de 100ms.

Configuration : Quartz à 6 Mhz et utilisation du timer0.

Solution :

1) Sélection du quartz en tant qu'horloge du timer 0 : => C/$\overline{T}$ = **0**

2) Contrôle du marche/arrêt en interne : => **GATE = 0**

3) La fréquence d'incrémentation du timer correspond à 6MHz/12 =500 kHz. 1/500kHz correspond à une période de 2µs. Il faudra compter 100.000µs/2µs impulsions d'horloge (soit 50.000) pour que le timer flag nous indique que 100.000 µs se sont écoulées.

4) Pour compter 50000 il faut configurer le TIMER en mode 1 (16 bits) soit M1=0 et M0=1.)

5) Comme le registre TMOD n'est pas accessible bit il faut calculer en hexadécimal la valeur d'initialisation de TMOD. Cela donne XXXX0001b soit (01h)

6) Il faut ensuite charger le complément à 2^{16} (65536) de 50000 soit 15536 (3CB0h), soit une initialisation de **TH0** à **3Ch** et de **TL0** à **B0h**

7) On peut s'assurer que le timer flag = 0 => **raz de TF0**

8) Il suffit de lancer le timer par le bit TR0 (timer run) => **TR0 = 1**

9) Ensuite on peut tester le passage à 1 du bit TF0 (timer flag)

Programme associé :

```
        mov  tmod,#01h      ; choix du mode
        mov  th0,#3Ch       ; initialisation du poids fort du timer
        mov  tl0,#0B0h      ; initialisation du poids faible du timer
        clr  tf0            ; raz du flag indiquant la fin de temporisation
        setb tr0            ; lancement du timer
attenteTF0:                 ; label de boucle d'attente
        jnb  tf0,attenteTF0 ; attente si le flag est toujours à 0
        clr  tf0            ; le timer flag 0 vient de passer à 1; raz du drapeau.
traitement:
; traitement correspondant à la fin de la temporisation
        |    |
        |    |
        |    |
```

IV.4. La liaison série

IV.4.1. Constitution

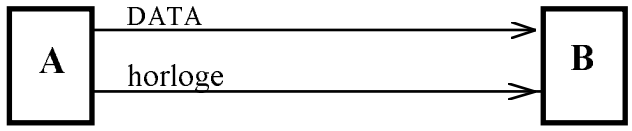
2 registres utilisés :

SBUF registre de stockage des données reçues ou à transmettre.

SCON registre de contrôle de fonctionnement de la liaison série.

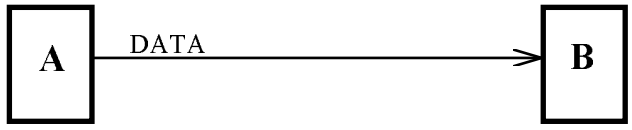
Principe : 2 types de transmission possibles

transmission synchrone



lors d'une transmission synchrone il faut transmettre les données ainsi que l'horloge donnant le rythme de changement des éléments binaires.

transmission asynchrone



lors d'une transmission asynchrone l'horloge donnant le rythme de positionnement des éléments binaires sur le support n'est pas transmise; ceci implique que l'émetteur ainsi que le récepteur doivent connaitre la vitesse de transmission (exprimée en bauds) et qu'en plus des éléments binaires de données il est nécessaire de transmettre des bits de synchronisation (bit de start et bit de stop).

Figure 20 : *Deux types de liaison série.*

La liaison série asynchrone est la plus utilisée.

Principe d'utilisation :

En émission :

Positionner par programme le flag appelé TI (indicateur de fin de transmission contenu dans SCON) à 0.

```
clr   TI
```

Ecriture dans SBUF

```
mov  SBUF,A    ; => début de la transmission automatique.
```

En fin de transmission le flag TI passe à 1 pour indiquer que le système d'émission est à nouveau disponible.

En réception :

Positionner par programme le bit REN (reception enable) à 1.

```
setb  REN      ; Aussitôt le système se met en attente de réception.
```

Positionner par programme le flag RI (indicateur de fin de réception) à 0

```
clr   RI
```

A la réception le flag RI se positionne à 1 indiquant qu'une information est disponible en lecture dans SBUF.

Il faut donc tester le bit RI :

```
jb ou jnb   RI
```

Lire le contenu de SBUF si RI = 1

```
mov  A,SBUF    ; par exemple.
```

Repositionner le flag RI à 0 (afin de se rendre compte d'une future réception).

```
clr   RI
```

IV.4.2.Etude des différents modes

4 modes possibles. Choix par 2 bits de SCON : SM0 et SM1

Attention : SM0 correspond à 2^1 et SM1 à 2^0

SM0	SM1	mode	type	données	vitesse
0	0	0	synchrone	8 bits	quartz/12
0	1	1	asynchrone	1 start, 8 bits, 1 stop	fonction du timer1
1	0	2	asynchrone	1 start, 9 bits, 1 stop	quartz/32 ou /64
1	1	3	asynchrone	1 start, 9 bits, 1 stop	fonction du timer1

Table 11 : *Quatre modes de liaison série.*

Mode Synchrone : SM0=0 et SM1=0.

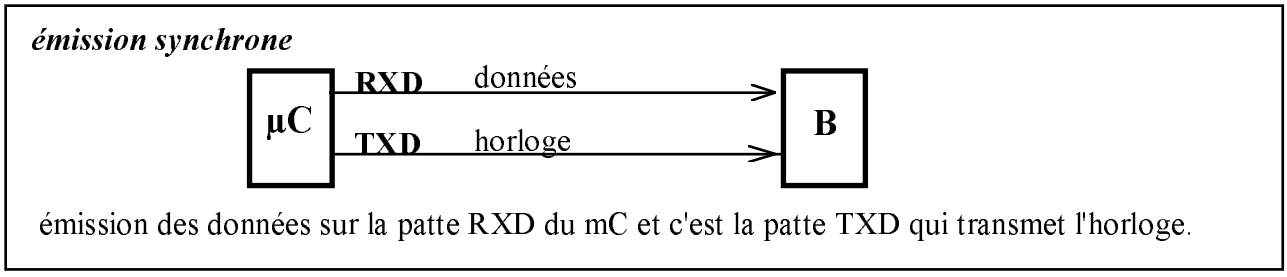
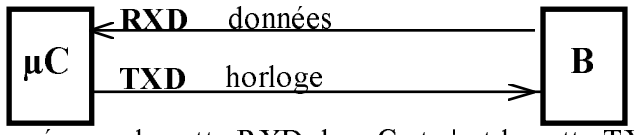


Figure 21 : *Emission série synchrone.*

réception synchrone



réception des données sur la patte RXD du mC et c'est la patte TXD qui transmet l'horloge.

Figure 22 : *Réception série asynchrone.*

8 bits en émission et en réception.

Impossible de recevoir et d'émettre en même temps.

Fréquence d'horloge fixe : Hqz/12.

C'est toujours le µC qui transmet l'horloge, en émission comme en réception => Utilisation difficile.

3 Modes asynchrones :

Principe

la réception des données se fait sur la patte RXD du µC et l'émission sur la patte TXD du µC

composition du signal asynchrone :

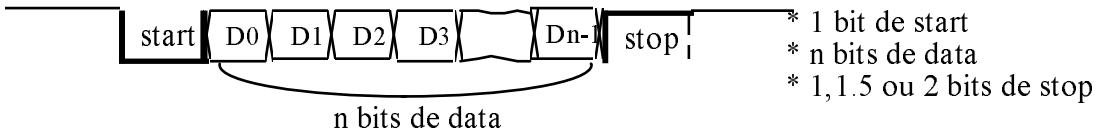


Figure 23 : *Principe de la liaison série asynchrone.*

Mode 1 : SM0=0, SM1=1.

Mode full duplex 10 bits

- 1 bit de start.
- 8 bits de data correspondant au contenu de SBUF.
- 1 bit de stop.

Dans ce mode la vitesse de transmission sera fonction du TIMER1 qui servira de diviseur de fréquences (voir vitesse de transmission pour les modes 1 et 3).

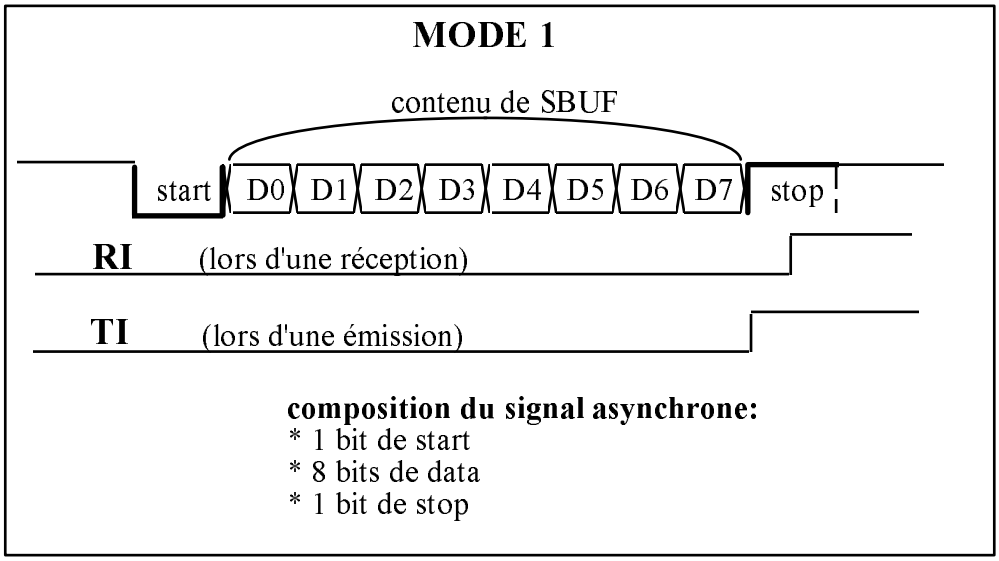


Figure 24 : *Liaison série asynchrone, mode 1.*

Mode 2 : SM0=1, SM1=0.

Mode full duplex 11 bits

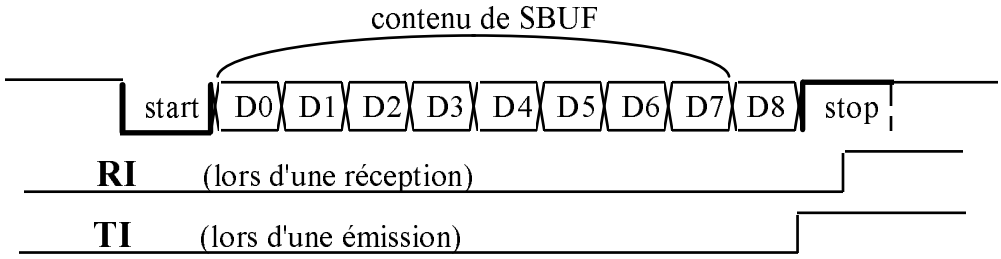
- 1 bit de start
- 9 bits de data:
  - 8 bits de données correspondants au contenu de SBUF.
  - +
  - 1 bit RB8 (contenu dans SCON) utilisé lors d'une réception.
  - ou
  - 1 bit TB8(contenu dans SCON) utilisé lors d'une émission.
- 1 bit de stop

2 horloges de transmission possibles en fonction d'un bit appelé SMOD (contenu dans PCON) :

SMOD = 0 => horloge de décalage de SBUF = Hqz/64

SMOD = 1 => horloge de décalage de SBUF = Hqz/32

MODE 2



composition du signal asynchrone :

* 1 bit de start
* 9 bits de data(SBUF + RB8 ou TB8) { D8 est copié dans RB8 lors d'une réception. c'est le contenu de TB8 qui constitue D8 le 9ème bit transmis.
* 1 bit de stop

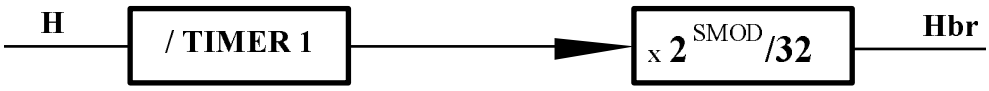
Figure 25 : *Liaison série asynchrone, mode 2.*

Mode 3 : SM0=1, SM1=1.

Equivalent au mode 2 avec l'horloge du mode 1.

IV.4.3.Vitesse de transmission en mode 1 ou 3

Principe :



H peut être égale à Hqz/12 ou à l'horloge qui arrive sur T1

Hbr (horloge de baud rate) est l'horloge qui provoque le décalage de SBUF

le TIMER est mis en cascade avec un autre compteur dont le modulo est 32 si le bit SMOD=0 ou bien 16 si le bit SMOD=1

$$\mathbf{H}br = \frac{\mathbf{H} \times 2^{\mathbf{SMOD}}}{\text{complément à 2 du TIMER1} \times 32}$$

Figure 26 : *Vitesse de transmission en mode 1 ou 3.*

Il faudra jouer sur la division H/TIMER1 afin d'obtenir la valeur souhaitée de **H**br. Quelque soit le mode utilisé pour le TIMER (13,16 ou 8 bits), il est peut probable que la division de **H** par 2^{13}, 2^{16} ou 2^{8} nous donne satisfaction; si nous désirons diviser **H** par *N (quelconque)* il faudra donc charger

le TIMER avec le complément à *N* du modulo maximum du TIMER, afin d'obtenir la division souhaitée de **H** .

Chaque fois que le TIMER1 passera par "0" il faudra recharger cette valeur correspondante au complément de *N*.. Soit cette opération se fera par logiciel en surveillant TF1, soit cette opération se fera automatiquement en utilisant le mode 2 (auto-reload) pour le fonctionnement du TIMER1. Dans ce cas il suffira d'initialiser correctement TH1.

Exemple :

H = hqz/12
quartz = 6Mhz
SMOD = 0
timer1 en mode 2 (autoreload donc sur 8 bits).

Nous avons donc $Hbr = \frac{6x106}{12 \times (256-TH1) \times 32}$

Si nous voulons, par exemple : Hbr = 1200 bauds

En posant V = 256 - TH1, nous avons $V = \frac{6 \times 106}{Hbr \times 12 \times 32} = 13,02$

Comme V doit être entier, nous avons V = 13 = 256 - TH1

Ceci implique que la valeur de chargement de TH1 = 243(en base 10) = 0F3h

IV.4.4.Registre de contrôle : SCON

9Fh	9Eh	9Dh	9Ch	9Bh	9Ah	99h	98h	adresse
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**	98h

Table 12 : *Registre SCON.*

RI fin de réception
TI fin d'émission
RB8 9ème bit reçu (mode 2 ou 3)
TB8 9ème bit transmis (mode 2 ou 3)
REN (receive enable) validation de la réception
SM2 validation protocole multiprocesseurs
SM0, SM1 choix du mode de transmission (SM0=2^1 et SM1=2^0)

IV.5. Les interruptions

IV.5.1. Schéma de principe

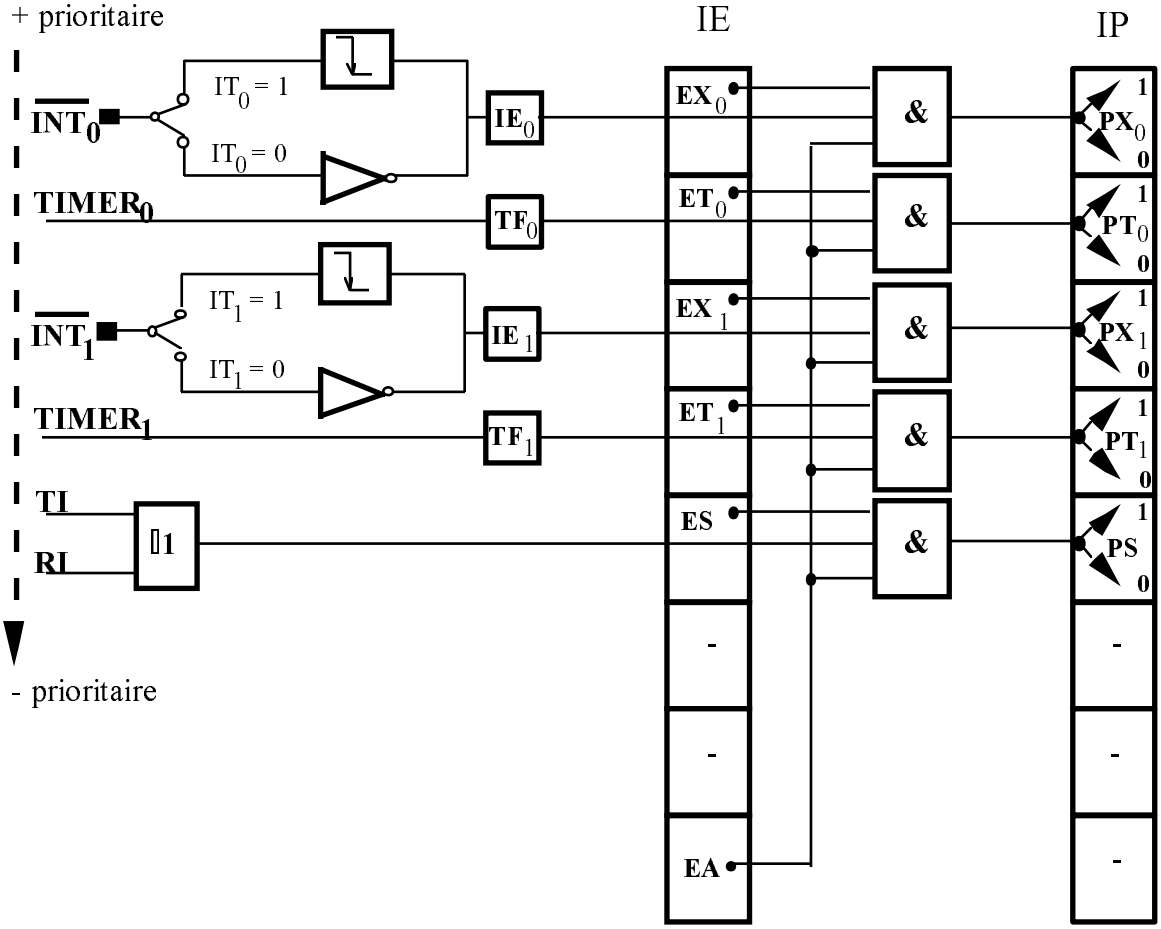


Figure 27 : *Schéma de principe des interruptions.*

Les sources de demande d'interruption sont au nombre de 5.

Chacune d'elles peut avoir 2 niveaux de priorité. Le choix du niveau se fait par l'intermédiaire du registre IP. Un 0 correspond à une faible priorité, un 1 correspond à une forte priorité.

A niveau égal dans IP c'est $\overline{\text{INT0}}$ qui est la plus prioritaire et la liaison série la moins prioritaire.

Pour valider une demande d'interruption il suffit de positionner à 1 le bit du registre IE correspondant à la demande et de positionner à 1 le bit EA (enable all, validation générale) du registre IE.

IV.5.2. Registres de gestion des interruptions

Registre IE (interrupt enable) : Registre de validation des interruptions

Accessible par l'adresse A8h (adressable bit A8h à AFh)

AFh	AEh	ADh	ACh	ABh	AAh	A9h	A8h	adresse
EA	**-**	**-**	**ES**	**ET1**	**EX1**	**ET0**	**EX0**	A8h

Table 13 : *Registre IE.*

Un bit de IE à "1" valide l'interruption. Un bit à "0" l'invalide.

EA (Enable All) : validation générale

ES (Enable Serial) : validation de la liaison série

ETx (Enable Timer_X) : validation du timerx

EXx (Enable External) : validation de la demande externe $\overline{INTx}$

Regisre IP (interrupt priority) : Registre de priorité des interruptions

Accessible par l'adresse B8h (adressable bit B8h à BFh)

BFh	BEh	BDh	BCh	BBh	BAh	B9h	B8h	adresse
-	-	-	**PS**	**PT1**	**PX1**	**PT0**	**PX0**	B8h

Table 14 : *Registre IP.*

Un "1" dans IP correspond à une forte priorité. Un "0" à une faible.

PS (Priority Serial) : Priorité de la liaison série.

PTx (Priority $Timer_X$) : Priorité du $timer_X$.

PXx (Priority External) : Priorité de la demande externe $\overline{INTx}$.

IV.5.3.Adresses de branchement lors d'une interruption

RESET :	Saut à l'adresse	0000h	de la mémoire
$\overline{INT0}$:	Saut à l'adresse	0003h	de la mémoire
TIMER0 :	saut à l'adresse	000Bh	de la mémoire
$\overline{INT1}$:	Saut à l'adresse	0013h	de la mémoire
TIMER1 :	Saut à l'adresse	001Bh	de la mémoire
TI ou RI :	Saut à l'adresse	0023h	de la mémoire

Comme nous pouvons le constater il y à très peu de place entre deux adresses pour pouvoir y placer un programme. Généralement nous placerons à ces adresses là des instructions de saut (LJMP) à l'adresse d'implantation du programme d'interruption.

Exemple :

```
        org   0000h     ; la prochaine instruction sera à l'adresse 0000h de la mémoire programme.
        ljmp  debut     ; saut à l'adresse du label debut
        org   00013h    ; la prochaine instruction sera à l'adresse 0013h de la mémoire programme.
        ljmp  pint1     ; saut à l'adresse du label pint1
        org   0030h     ; la prochaine instruction sera à l'adresse 0030h de la mémoire programme.
debut:  clr PX1         ; priorité de l'interruption INT1 à 0
        clr IE1         ; indicateur d'interruption : en attente
        setb EX1        ; demasque l'interruption INT1
        setb EA         ; validation générale des interruptions
        …               ; suite des initialisations
Princ:  …               ; programme principal
        ljmp Princ      ; boucle du programme principal
pint1:  push PSW        ; sauvegarde du contexte
        push ACC
        …               ; acquittement de l'iterruption
        …               ; traitement
        pop ACC         ; restitution du contexte
        pop PSW
        reti            ; retour au programme principal
```

Le RESET est l'interruption la plus prioritaire non masquable(on ne peut pas l'empêcher de se dérouler). De plus le RESET modifie les registres internes :

PC	program counter	0000h	(normal)

ACC	accumulateur	00h
B	accu auxiliaire	00h
PSW	program status word	00h
SP	stack pointer	07h (attention pile dans banques)
DPTR	data pointer register	0000h
P0..P3	Port0 à Port3	FFh
IP	interrup priority	XXX00000b
IE	interrup enable	0XX00000b
TMOD	timer mode	00h
TCON	timer control	00h
TH0	timer high n° 0	00h
TL0	timer low n° 0	00h
TH1	timer high n° 1	00h
TL1	timer low n° 1	00h
SCON	serial control	00h
SBUF	serial buffer	XXXXXXXXb
PCON	power control	0XXXXX00b

h = hexadécimal
b = binaire

La RAM interne n'est pas affectée par un RESET si l'alimentation du µC est maintenue.