10

Description des timers

Généralités

Chaque timer est doté d'un compteur 16 bits accessible sous la forme de deux registres de 8 bits faisant partie des registres à fonction spéciale ·

— TH0 et TL0 pour le timer 0,

TH1 et TL1 pour le timer 1.

Ces deux timers sont configurables à partir des deux registres TMOD et TCON.

Un timer peut assurer deux fonctions distinctes

Le COMPTAGE, le compteur est alors incrémenté par la détection d'événements extérieurs. Ces événements sont des changements d'états appliqués aux broches dédiées à cette fonction (T0 = P3.4 ou T1 = P3.5).

— La TEMPORISATION, le compteur est incrémenté à partir du signal d'horloge du microcontrôleur soit directement soit à travers un circuit de division de la fréquence.

En plus du choix entre ces deux fonctions, les timers 0 et 1 connaissent quatre modes de fonctionnement différents, permettant de modifier le format du registre de comptage ou d'autoriser un rechargement automatique à partir d'une valeur de consigne. Le timer 1 peut faire office de générateur de fréquences de communication de l'interface de communication série.

La configuration d'un timer (choix de sa fonction et de son mode) est effectuée par programmation du registre de contrôle TMOD.

TMOD : Registre de contrôle du mode

Le tableau 10.1 précise le rôle de chaque bit du registre de contrôle du mode de fonctionnement des timers 0 et 1. Quatre bits sont dédiés à chaque timer

TMOD adresse directe 89 H

GATE	C/T	M1	M0	GATE	C/T	M1	M0
TIMER 1				TIMER 0			

GATE GATE = 1, le TIMER x est validé seulement lorsque sa broche d'entrée "INTx" est à l'état 1 et que le bit "TRx" du registre TCON est à 1.
GATE = 0, le TIMER est validé si "TRx" = 1.

C/T Permet la sélection de la fonction du TIMER
C/T = 0 le TIMER est en fonction TEMPORISATEUR
C/T = 1 le TIMER est en fonction COMPTEUR à partir des événements présents sur la broche d'entrée "Tx"

M1	M0	Mode
0	0	compteur 13 bits compatible au microcontrôleur MCS-48
0	1	compteur 16 bits
1	0	compteur 8 bits à rechargement automatique. Le contenu de THx est rechargé dans TLx lorsque celui-ci repasse à 0.
1	1	TIMER 0 Le registre TL0 devient un compteur 8 bits contrôlé normalement par les bits de contrôle du TIMER 0. Par contre, THO est configuré en temporisateur 8 bits contrôlé par les bits de contrôle du TIMER 1.
1	1	TIMER 1 compteur 1 à l'arrêt.

Tableau 10.1 : Détail du registre de contrôle TMOD

Le bit GATE permet de valider la prise en compte des changements d'état ou des transitions détectées sur la broche externe INT0 ou INT1 (selon le timer utilisé). Le bit C/T permet de choisir entre la fonction "*compteur*" et la fonction "*temporisateur*" Les deux bits M0 et M1 assurent le choix parmi l'un des quatre modes de fonctionnement possibles.

C/T : Choix de la fonction du timer

Selon la valeur des bits C/T du registre TMOD, la fonction de chaque timer peut être modifiée.

Changer la fonction d'un timer consiste à changer la source des événements qui incrémentent le compteur du timer

Fonction temporisation

Dans cette fonction, le compteur est incrémenté automatiquement à chaque cycle machine. Un indicateur est positionné lorsque le compteur, par phénomène de débordement, (*overflow*) repasse à la valeur 0. Ces indicateurs apparaissent dans le registre TCON (figure 10.6, page 120) sous les noms TF0 et TF1.

Fonction compteur

Le registre du timer est incrémenté sur une transition négative prise en compte à partir de l'entrée du timer (T0, T1, ou T2). Dans ce cas la broche d'entrée est échantillonnée durant la phase S5P2 de chaque cycle machine.

Lorsque l'échantillonneur "voit" un état haut puis, lors du cycle suivant, un état bas, le compteur est incrémenté. La nouvelle valeur apparaît dans le registre durant la phase S3P1 du cycle suivant la détection de la transition. Il faut donc deux cycles machines pour détecter et enregistrer un événement. Cette constatation limite la fréquence de comptage à une valeur vingt quatre fois moins importante que celle de l'oscillateur du microcontrôleur

Description du mode opératoire (M1 et M0)

Le choix du mode est obtenu par position des bits M0 et M1 du registre TMOD.

Mode 0

Ce mode assure un fonctionnement des timers compatible à celui des timers des microcontrôleurs de la famille MCS-48. L'organisation fonctionnelle de ce mode est donnée figure 10.2. Le timer fonctionne alors en compteur 8 bits précédé par un prédiviseur de 32. Le registre du compteur est donc configuré en registre 13 bits 5 bits sur la partie poids faible pour le prédiviseur et 8 bits sur poids fort pour le comptage. Lorsque le compteur repasse à 0, par débordement, l'indicateur d'interruption TFx est mis à 1. L'entrée de comptage est validée lorsque TRx (bit de contrôle du registre TCON) est à 1 et que GATE (bit de mode dans TMOD) est à 0 ou, lorsque l'entrée INTx présente un état 1.

Mode 1

Le mode 1 est identique au mode 0 excepté le fait que le registre du compteur utilise la pleine capacité des 16 bits sans prédiviseur L'organisation fonctionnelle de ce mode est représentée par la figure 10.3.

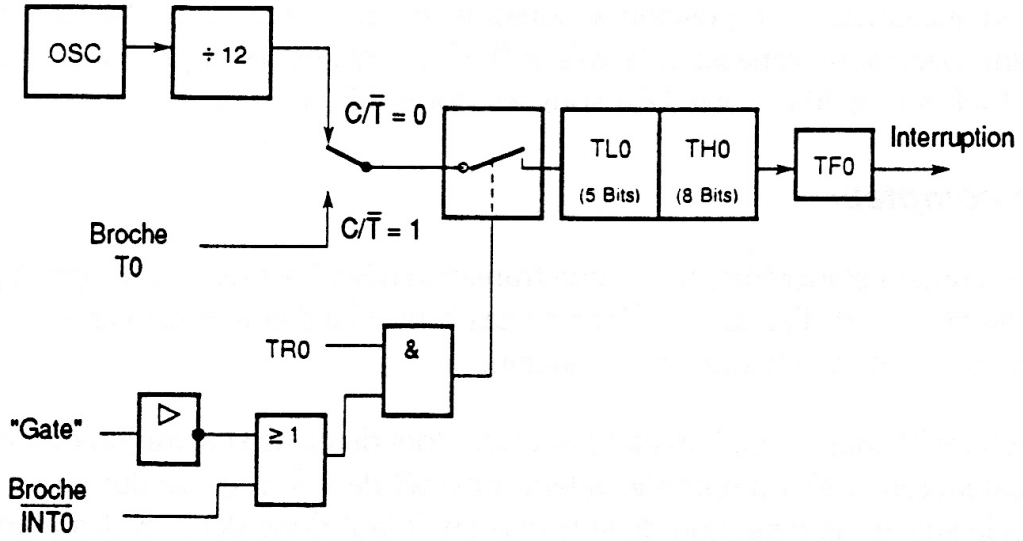


Figure 10.2 : Timer 0 ou 1 en mode 0 ***(d'après document Siemens)***

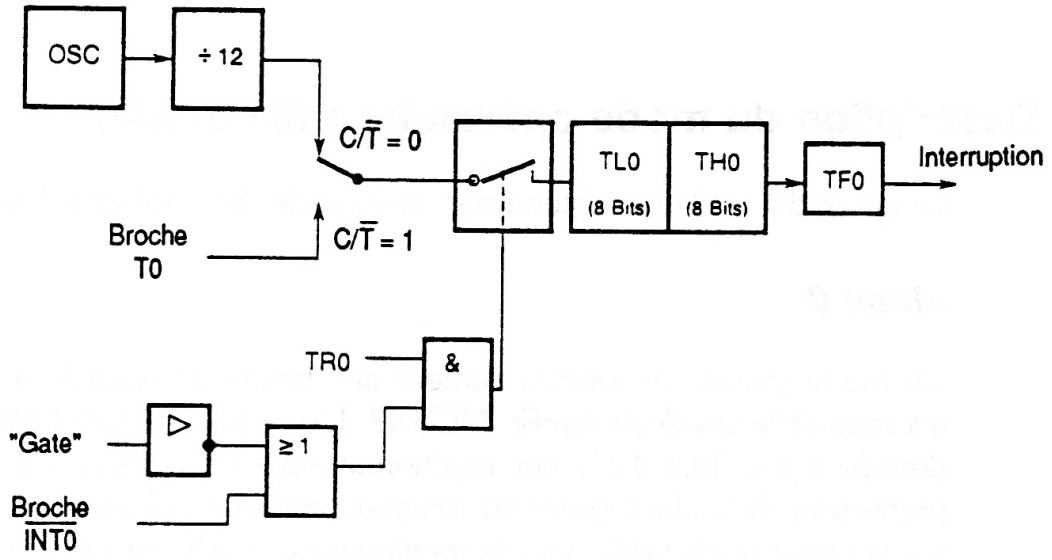


Figure 10.3 : Timer 0 ou 1 en mode 1 ***(d'après document Siemens)***

Mode 2

Dans ce mode le registre du timer est configuré en un compteur 8 bits à rechargement automatique (figure 10.4). La partie de poids faible TLx (TL0 ou TL1 selon le timer utilisé) sert de compteur Lorsque ce compteur repasse à 0 par débordement, l'indicateur TFx est mis à 1 et le contenu de THx est rechargé dans le compteur TLx. Le contenu de THx, déterminé par programmation reste inchangé.

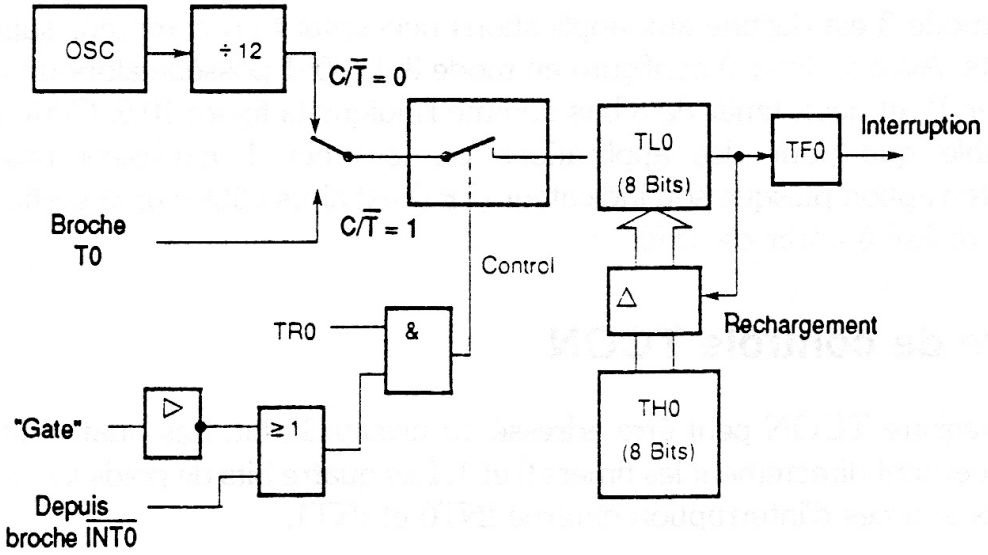


Figure 10.4 Timer 0 ou 1 en mode 2

Mode 3

Le timer 1 en mode 3 est bloqué comme si son bit de contrôle TR1 était à 0.

Par contre pour le timer 0 (figure 10.5), les registres TL0 et TH0 sont configurés comme 2 compteurs de 8 bits. Le compteur TL0 utilise la logique de contrôle du timer 0 (C/T, GATE, TR0, INTO, et TF0) alors que le compteur TH0 est bloqué en fonction temporisateur sous le contrôle de TR1. Le compteur TH0 positionne l'indicateur TF1, et récupère donc la fonction d'interruption du timer 1.

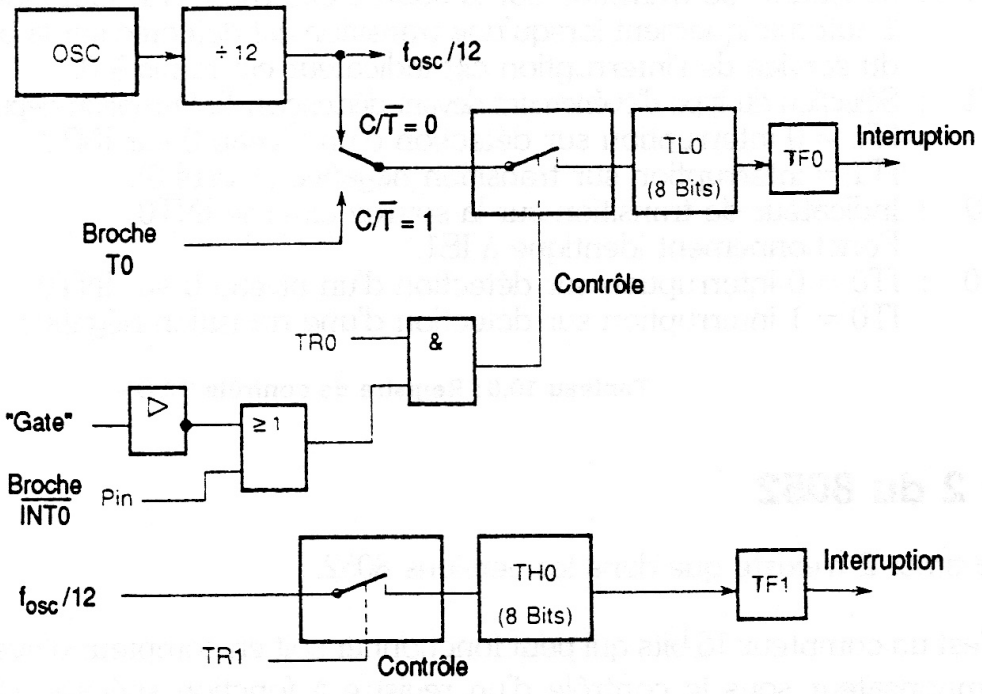


Figure 10.5 : Timer 0 en mode 3 : 2 compteurs 8 bits

Le mode 3 est destiné aux applications nécessitant un compteur supplémentaire de 8 bits. Avec le timer 0 configuré en mode 3, le 8051 possède alors un timer 16 bits (le timer 1), et deux timer de 8 bits comme l'indique la figure 10.5. Cette remarque n'est valable que pour des applications où le timer 1 n'utilisera pas sa ressource d'interruption puisque son indicateur (TF1) est alors câblé sur la sortie du compteur 8 bits réalisé à partir de TH0.

Registre de contrôle TCON

Le registre TCON peut être adressé au niveau du bit. Les quatre bits de poids fort concernent directement les timers 0 et 1. Les quatre bits de poids faible concernent les deux sources d'interruption externe INT0 et INT1.

TCON Adresse directe 88 H adressable au niveau bit

8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 Indicateur de débordement du TIMER 1. Mis à 1 automatiquement lorsque le compteur repasse à 0. Si l'interruption correspondante est autorisée, cet indicateur est remis à 0 automatiquement lorsque le sous-programme d'interruption est exécuté.

TR1 Bit de déclenchement du TIMER 1. Il doit être positionné par logiciel à 1 (pour lancer) ou 0 (pour arrêter) le TIMER.

TF0 Indicateur de débordement du TIMER 0. Identique à TF1.

TR0 Bit de déclenchement du TIMER 0. même fonctionnement que TR1.

IE1 Indicateur de transition sur la source externe INT1. Cet indicateur est mis à 1 automatiquement lorsqu'une transition est détectée sur la borne INT1. Lors du service de l'interruption cet indicateur est remis à 0.

IT1 Sélection du type d'événement devant déclencher l'interruption depuis la source INT1.
IT1 = 0 interruption sur détection d'un niveau 0 sur INT1.
IT1 = interruption sur transition négative (1 vers 0).

IE0 Indicateur de transition sur la source externe INT0.
Fonctionnement identique à IE1.

IT0 IT0 = 0 interruption sur détection d'un niveau 0 sur INT0.
IT0 = 1 interruption sur détection d'une transition négative.

Tableau 10.6 : Registre de contrôle TCON

Timer 2 du 8052

Le timer 2 n'existe que dans les versions 8052.

C'est un compteur 16 bits qui peut fonctionner soit en compteur d'événements, soit en temporisateur sous le contrôle d'un registre à fonction spéciale T2CON d'adresse directe C8H. Trois modes de fonctionnement sont possibles. Il s'agit des modes

"capture", "rechargement", et générateur de fréquence pour communication série. Ce dernier mode de fonctionnement utilise une définition sur 16 bits. L'utilisation du timer 2 en remplacement du timer 1 permet donc un choix plus grand de vitesse de communication sur le port série.

Registre T2CON

T2CON adresse directe C8H adressable au niveau bit

CFH	CEH	CDH	CCH	CBH	CAH	C9H	C8H
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C / $\overline{T2}$	CP/$\overline{RL2}$

TF2: *Dépassement de capacité. Mis à 1 lorsque le Timer 2 passe de la valeur FFFFH à la valeur 0.*

EXF2: *Indicateur externe. Mis à 1 lorsqu'une capture ou un rechargement est provoqué par transition 0 vers 1 sur broche T2EX et que EXEN2 = 1. Si l'interruption du Timer 2 est autorisée et que EXF2=1 alors le sous-programme placé à l'adresse 2BH est exécuté. La remise à 0 de EXF2 doit être assurée par programmation.*

RCLK: *Bit de contrôle de l'horloge de réception. RCLK =1 positionné à 1, le port série, programmé en mode 1 ou 3, utilise l'impulsion de dépassement du Timer 2 comme signal d'horloge de réception. RCLK =0, c'est l'impulsion de dépassement du Timer 1 qui joue ce rôle.*

TCLK. *Contrôle de l'horloge d'émission. TCLK = 1, le port série, (en mode 1 ou 3), utilise l'impulsion de dépassement du Timer 2 comme signal d'horloge d'émission. TCLK = 0, c'est l'impulsion de dépassement du Timer 1 qui joue ce rôle.*

EXEN2: *Validation du mode externe. EXEN2 = 1 autorisation de capture ou de rechargement provoqué par une transition négative détectée sur broche T2EX. Ceci n'est possible que si le Timer 2 n'est pas utilisé comme horloge du port série (TCLK et RCLK à 0). Le fait de forcer EXEN2 à 0 permet d'ignorer les événements présents sur la broche T2EX.*

TR2 : *Contrôle du fonctionnement. La mise à 1 de TR2 provoque le démarrage du Timer*

C/$\overline{T2}$: *Sélection du mode de fonctionnement du Timer 2*

C / $\overline{T2}$=0 fonction "temporisateur",

C / $\overline{T2}$=1 fonction "compteur d'événements externes"

En mode temporisateur, c'est l'horloge (du microcontrôleur) divisée par 12 qui commande l'incrémentation du Timer En mode compteur, seules des transitions négatives peuvent être détectées comme événements externes.

CP/$\overline{RL2}$: *Sélection pour capture ou rechargement. CP/$\overline{RL2}$ = 1 le mode capture est sélectionné. La conjugaison EXEN2=1 et détection d'une transition négative sur T2EX permet une capture la valeur contenue par le registre 16 bits du Timer 2 est placée dans les registres RCAP2H et RCAP2L. Ce mode de capture n'est possible que si RCLK et TCLK sont à 0. Dans le cas contraire, c'est le mode rechargement automatique qui est sélectionné.*

Fonctionnement en mode capture

Illustré par la figure 10.8, ce mode permet de "capturer" ou mémoriser la valeur courante du timer 2. Le timer 2, est un compteur de 16 bits composé des registres TH2 et TL2. L'opération de capture recopie la valeur de TH2 dans le registre RCAP2H et la valeur de TL2 dans RCAP2L.

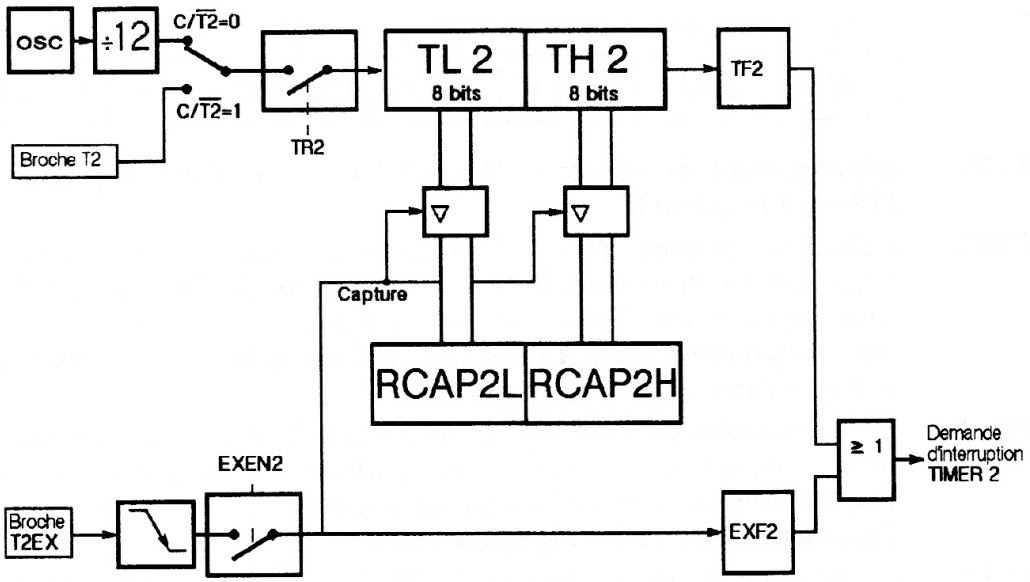


Figure 10.8 Timer 2 en mode capture

Dans le mode "capture" l'état du bit EXEN2 détermine deux options de fonctionnement.

EXEN2 = 0 le timer 2 peut alors fonctionner en compteur ou en temporisateur 16 bits. Lors du dépassement de capacité du registre 16 bits, le bit TF2 est automatiquement positionné à 1, ce qui peut produire une interruption (à condition qu'elle soit autorisée par le programmeur).

EXEN2 = 1 le fonctionnement général reste identique à celui qui vient d'être décrit. La différence réside dans la prise en compte d'événements extérieurs. Lors d'une transition logique 1 vers 0 (transition négative) sur la broche T2EX les contenus des registres TH2 et TL2 sont copiés respectivement dans les registres RCAP2H et RCAP2L. Dans le même temps, l'indicateur EXF2 est positionné à 1 et peut ainsi permettre le déclenchement d'une interruption. Il faut remarquer que les indicateurs TF2 et EXF2 sont affectés du même vecteur d'interruption (adresse 002BH). Lors du traitement d'interruption, le programmeur devra déterminer par test, l'événement ayant provoqué l'interruption.

Fonctionnement en "rechargement automatique"

Comme pour le mode "capture", ce mode de fonctionnement est contrôlé par le bit EXEN2.

EXEN2 = 0 le compteur évolue jusqu'au dépassement de capacité, et positionne l'indicateur TF2, le compteur, c'est-à-dire TH2 et TL2, est alors rechargé de la valeur contenue par RCAP2H et par RCAP2L. Ces deux derniers registres doivent bien sûr, avoir été initialisés d'une donnée par le programmeur Ce mode de fonctionnement peut être utilisé comme générateur de fréquence pour la communication série soit seulement pour la réception (RCLK = 1), soit seulement pour l'émission (TCLK = 1), ou encore pour les deux (RCLK = 1 et TCLK = 1).
EXEN2 = 1 :le fonctionnement en rechargement automatique reste similaire. En plus, une transition négative sur la broche T2EX provoque un rechargement anticipé et provoque la mise à 1 de EXF2.

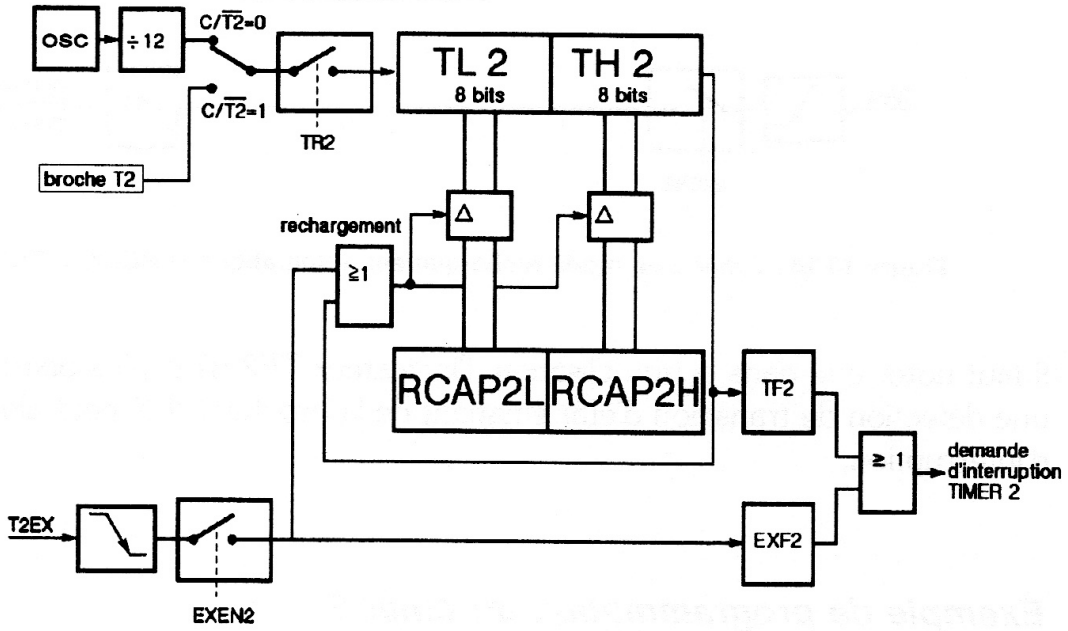


Figure 10.9 Timer 2 en mode rechargement automatique

Lorsque le mode "*rechargement automatique*" est utilisé comme générateur de fréquence pour la communication série (RCLK+TCLK = 1), les sous-ensembles du timer 2 prennent l'organisation précisée par la figure 10.10. Il faut remarquer que le compteur est alors commandé par une fréquence plus élevée ($F_{OSC}/2$ au lieu de $F_{OSC}/12$). La fréquence de communication FC est donnée par la formule

$$FC = \frac{F_{OSC}}{32.\ [65536 - (RCAP2H, RCAP2L)]}$$

où RCAP2H et RCAP2L représentent la valeur 16 bits de rechargement du compteur

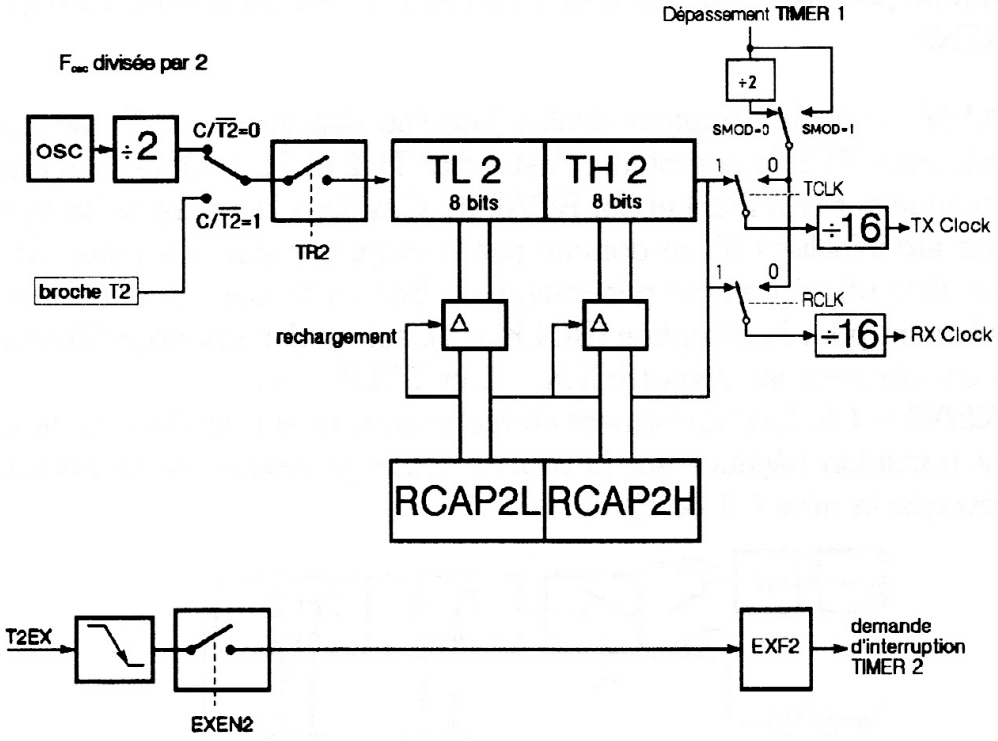


Figure 10.10 : Timer 2 en mode rechargement automatique si RCLK + TCLKG = 1

Il faut noter que dans cette utilisation, l'indicateur TF2 n'est plus opérationnel. Seule une détection de transition d'état émanant de la broche T2EX peut alors être source d'interruption.

Exemple de programmation du timer 2

A titre d'exemple, on se propose de décrire le module d'initialisation du timer 2 tel que, celui-ci provoque une interruption toutes les millisecondes. Le programme d'interruption, non décrit, sera placé à partir de l'adresse 002BH.

Le mode "rechargement automatique" est retenu pour cette application. Il faut donc définir les valeurs à placer dans les registres RCAP2H et RCAP2L pour obtenir une temporisation d'une milliseconde. Le compteur est incrémenté tous les cycles machines, soit toutes les microsecondes si le quartz de l'oscillateur est de 12 MHz. Ce compteur provoque l'interruption lorsque son contenu dépasse la valeur 65 535. Il faut donc partir de la valeur

$$65\,536 \quad 1\,000 = 64\,536$$

Cette valeur de 16 bits est à répartir entre le registre de poids fort RCAP2H et le registre de poids faible RCAP2L suivant la formule suivante

(RCAP2H x 256)+(RCAP2L) = 64 536

contenu de RCAP2H 64 536/256 = 252

contenu de RCAP2L 64536 (256 x 252) = 24

Pour assurer une première période d'une milliseconde, il est souhaitable d'initialiser TH2 et TL2 de ces mêmes valeurs.

Une fois les registres initialisés, il faut déclencher le temporisateur en mettant à 1 le bit TR2 et autoriser la source d'interruption du timer 2.

INITIALISATION DU TIMER 2

MODE RECHARGEMENT AUTOMATIQUE

65 536-(RCAP2H x 256+RCAP2L)= 1 ms

```
MOV     RCAP2H,#252          ;252 x 256
MOV     RCAP2L,#24           ;+24
MOV     TH2,#252
MOV     TL2,#24
MOV     T2CON,#00000100B     ;LANCER TIMER TR2 = 1
MOV     IE,#10100000B        ;AUTORISER IT TIMER 2
RET
```

Générateur de fréquence de communication série

Utilisation du timer 1

Le timer 1 peut être utilisé comme horloge pour la communication série. Il doit être configuré alors en mode 2, c'est-à-dire en compteur huit bits à rechargement automatique. Dès que le registre TL1 repasse à 0, il est rechargé de la valeur contenue dans le registre TH1. La vitesse de communication FC est définie par la formule suivante

$$FC = \frac{K \times f_{OSC}}{32 \times 12(256 \quad TH1)}$$

où f_{OSC} = fréquence de l'oscillateur du microcontrôleur
et K = coefficient de valeur 1 ou 2.

La valeur de K est définie par l'état du bit SMOD du registre PCON

si SMOD = 0, alors $K = 1$,
si SMOD = 1, alors $K = 2$.

D'une façon pratique, il est nécessaire de définir la valeur à placer dans le registre TH1 en fonction d'une fréquence FC. La formule devient

$$TH1 = 256 \quad \frac{K \times f_{OSC}}{384 \times FC}$$

Exemple de calcul On se propose de définir la valeur de TH1 pour obtenir une vitesse de communication normalisée de 19 200 Bauds avec une tolérance de 2 %. La fréquence de l'oscillateur est de 12 MHz et K est choisi comme égal à 2. Il faut placer dans TH1 la valeur théorique de

$$256 \quad \frac{2 \times 12 \times 10^6}{384 \times 19\,200} = 252{,}744\,79$$

La valeur à placer dans TH1 ne peut être qu'un entier huit bits. En optant pour une valeur de 252 par défaut, le calcul de vérification de FC conduit à une fréquence de 15 625 Bauds. Par contre le calcul de FC pour la valeur par excès de TH1 = 253 donne une fréquence de 20 833 Bauds. Ces deux valeurs ne sont pas comprises dans la tolérance fixée.

Cet exemple fait donc apparaître l'impossibilité d'obtenir, à partir du timer 1, une vitesse de communication de 19 200 Bauds avec un quartz de 12 MHz comme base de temps pour l'oscillateur

Le constructeur INTEL propose l'utilisation d'un quartz de 11,059 MHz pour résoudre ce problème. Un nouveau calcul avec cette valeur de quartz donne

$$256 - \frac{2 \times 11\,059 \times 10^6}{384 \times 19\,200} = 253{,}00005$$

En tenant compte de la tolérance de 2 %, ce calcul reste valable pour tout quartz dont la fréquence d'oscillation serait comprise entre 10,9 MHz et 11,3 MHz. Un quartz de 11 MHz peut donc parfaitement convenir.

Dans l'exemple de calcul, il a été choisi un coefficient K = 2, ce qui correspond à un état logique 1 du bit SMOD. Ce bit étant initialisé à l'état 0 par le microcontrôleur, il est donc nécessaire de positionner ce bit dans le registre PCON qui n'est accessible qu'en adressage direct. L'instruction

ORL PCON,#80H

positionne à 1 le bit de poids fort (SMOD) de PCON situé à l'adresse 87H

Le programme complet d'initialisation du timer 1, fonctionnant comme générateur de vitesse de communication, est le suivant

Configuration du timer 1

en mode 2 rechargement automatique de TL1 par TH1

TH1 = 253

lancé lorsque TR1 = 1 (TR1 bit 6 de TCON)

Définition de TMOD

TIMER 0				TIMER 1			
GATE	C/T	M1	M0	GATE	C/T	M1	M0
0	0	1	0	0	0	0	0

soit la valeur 20H (le timer 0 n'étant pas pris en compte).

```
MOV     TMOD,#20H     , timer 1 en MODE 2
MOV     TH1,#253      , fréquence de 19200 bauds
ORL     PCON,#80H     , avec SMOD = 1 pour K = 2
SETB    TR1           et lancer le timer
```

Le tableau 10.11 donne les valeurs à affecter à TH1 pour obtenir certaines vitesses de communication standard utilisée par exemple par la norme RS 232 ou la norme MIDI (communication entre instruments musicaux).

Fréquence de communication	Fréquence de l'oscillateur	Valeur de SMOD	C/T	TIMER 1	
				Mode	TH1
19200	11.059 MHz	1	0	2	FDH
9600	11.059 MHz	0	0	2	FDH
4800	11.059 MHz	0	0	2	FAH
2400	11.059 MHz	0	0	2	F4H
1200	11.059 MHz	0	0	2	E8H
300	6 MHz	0	0	2	CCH
110	6 MHz	0	0	2	72H
MIDI					
31250	12 MHz	0	0	2	FFH

Tableau 10.11 : Fréquence de communication à partir du TIMER 1

Utilisation du timer 2 du 8052

Lorsqu'on utilise le timer 1 en mode "rechargement automatique", le réglage de la fréquence est limité à une résolution de 8 bits. Cette limitation a été mise en évidence dans l'exemple précédent une horloge de 12 MHz ne permettait pas d'obtenir une fréquence de communication de 19 200 bauds à 2 % près.

Le timer 2, lorsqu'il est utilisé en mode "rechargement automatique" (voir page 123), fonctionne sur 16 bits. Le timer 1 peut être remplacé partiellement ou totalement par le timer 2 dans la fonction d'horloge du port série. Ce remplacement est programmé à partir des bits RCLK et TCLK du registre T2CON

RCLK	TCLK	Fonction
0	0	le timer 1 contrôle le port série.
0	1	le timer 2 contrôle la vitesse d'émission et le timer 1 contrôle la vitesse de réception.
1	0	le timer 2 contrôle la vitesse de réception et le timer 1 contrôle l'émission.
1	1	le timer 2 contrôle émission et réception.

Pour le timer 2, la fréquence de communication est donnée par la formule

$$FC = \frac{f_{OSC}}{32 \times [65\,536 \quad (\text{RCAP2H, RCAP2L})]}$$

où RCAP2H,RCAP2L représente un entier de 16 bits.

Ces deux registres servent au rechargement de TH2 et de TL2.

De cette formule on peut définir une formule pratique qui permettra de définir les valeurs à placer dans RCAP2H et RCAP2L pour une fréquence de communication choisie

$$(\text{RCAP2H, RCAP2L}) = 65\,536 \quad \frac{f_{OSC}}{32 \times FC}$$

A partir de cette formule, déterminons les contenus de RCAP2H et RCAP2L pour obtenir une fréquence de communication FC de 19 200 bauds si F_{OSC} est égale à 12 MHz

$$65\,536 \quad \frac{12 \times 10^6}{32 \times 19\,200} = 65\,516{,}469$$

soit la valeur entière de 65516 ou FFECH. Il faut donc placer la valeur FFH dans RCAP2H et la valeur ECH dans RCAP2L. Avec la valeur arrondie de FFECH la valeur de FC est de 18750 Bauds. Cette valeur est très proche de la tolérance inférieure à 2 % (18816 Bauds). En toute rigueur, FC reste en dehors de la tolérance fixée, même si la valeur obtenue reste plus acceptable que la valeur obtenue avec le timer 1.

Le tableau 10.12 donne des valeurs standards de FC et les valeurs à placer dans RCAP2H et RCAP2L pour les obtenir

Fréquence de communication	Fréquence de l'oscillateur	TIMER 2	
		RCAP2H	RCAP2L
9600	12 MHz	FF	D9
4800	12 MHz	FF	B2
2400	12 MHz	FF	64
1200	12 MHz	FE	C8
300	12 MHz	FB	1E
110	12 MHz	F2	AF
300	6 MHz	FD	8F
110	6 MHz	F9	57

Tableau 10.12 Fréquence de communication à partir du TIMER 2

Le programme suivant est un exemple d'initialisation de communication par le port série. La vitesse de communication est la même en émission et en réception 9600 Bauds. Le port série est initialisé en mode 1 (voir chapitre suivant)

```
MOV     SCON, #01011010B  , initialise port série
MOV     RCAP2H, #0FFH     , valeur de rechargement pour TH2
MOV     RCAP2L, #0D9H     , valeur pour TL2
MOV     T2CON, #34H       , TCLK et RCLK à 1 et lancer timer
```

Un autre exemple où la réception est pilotée par le timer 1 à une vitesse de 4 800 bauds alors que la vitesse de l'émission de 2 400 bauds est obtenue grâce au timer 2

```
MOV     SCON, #01011010B  , initialise port série
MOV     RCAP2H, #0FFH     , valeur de rechargement pour TH2
MOV     RCAP2L, #064H     , valeur pour TL2
MOV     T2CON, #00010100B , TCLK à 1 et lancer timer 2
MOV     TMOD, #00100000B  , 8 bits auto reload timer 1
MOV     TH1, #0FAH        , valeur pour 4800 bauds
SETB    TR1               , lancer timer 1
```

11

Interface de communication série

Présentation

L'interface de communication série est matériellement accessible par la fonction secondaire de deux lignes du port P3. Il s'agit de la borne RXD (fonction secondaire de P3.0) pour la réception, et de la borne TXD (deuxième fonction de P3.1).

Le port série fonctionne en *"full duplex"*, c'est-à-dire en réception et transmission simultanée. Il possède aussi un tampon de réception qui autorise la réception d'un octet avant même la lecture du précédent. Cependant la lecture du premier octet doit être effectuée avant la réception du dernier bit du deuxième octet. Les registres de réception et de transmission sont tous deux accessibles, à la même adresse de la RAM interne, désigné par le symbole SBUF dont la valeur est 99H. Une écriture à cette adresse provoque un accès au registre de transmission, alors qu'une lecture du contenu de cette adresse permet de récupérer la donnée du registre de réception.

La commande et le contrôle du port série s'effectuent grâce au registre à fonction spéciale SCON situé à l'adresse 98H. Le port série peut fonctionner selon quatre modes ·

MODE 0 Registre à décalage
Le port série est utilisé comme un registre à décalage au format de 8 bits. Les données série entrent ou sortent par la broche RXD La broche TXD est utilisée comme horloge dont la fréquence est obtenue par division par 12 de la fréquence de l'oscillateur du microcontrôleur

MODE 1 Le port série fonctionne en UART 8 bits avec un bit de départ et un bit de stop. 10 bits sont ainsi transmis (par la borne TXD) ou reçus (sur la borne RXD). En réception, le bit de STOP est placé dans le registre SCON. La vitesse de communication est variable. Elle est définie par programmation du TIMER 1 pour le 8051 et des TIMER 1 et 2 pour le 8052. Cette particularité autorise alors des vitesses différentes pour la réception et la transmission.

MODE 2 Le port série est configuré en UART de format 9 bits. La donnée est encadrée d'un bit de start et d'un bit de stop. Pour la transmission, le neuvième bit provient du registre de contrôle SCON, en réception ce neuvième bit sera placé dans ce même registre. La vitesse d'émission/réception est programmable soit 1/32 ou 1/64 de la fréquence oscillateur du microcontrôleur

MODE 3 Le mode 3 est identique au mode 2. La seule différence réside dans le fait que la vitesse d'émission/réception est variable elle est générée par le compteur 1 (1 ou 2 sur le 8052).

Registre de contrôle

Le registre de fonction spéciale SCON assure le contrôle et indique l'état de l'interface série. SCON permet entre autre, la sélection du mode de communication.

SCON Adresse directe 98 H adressable au niveau du bit

9F	9E	9D	9C	9B	9A	99	98
SMO	SM1	SM2	REN	TB8	RB8	TI	RI

mode de fonctionnement du port série

SM0	SM1	Mode	Description	fréquence de communication
0	0	0	registre à décalage	fosc./12
0	1	1	UART 8 bits	variable
1	0	2	UART 9 bits	fosc./64 ou fosc./32
1	1	3	UART 9 bits	variable

SM2 Communication multiprocesseur possible en mode 2 et 3.
En mode 2 ou 3, si SM2=1, alors RI ne sera activé que si le 9^{eme} bit est un 1.
En mode 1, si SM2 = 1, alors RI ne sera activé qu'à la réception d'un bit de stop valide.
En mode 0, SM2 doit être positionné à 0.

REN Mis à 1 par programme REN autorise la réception série.

TB8 correspond au 9^{eme} bit à transmettre en mode 2 et 3.

RB8 correspond au 9^{eme} bit reçu en mode 2 et 3. En mode 1, si SM2 = 0, RB8 correspond au bit de stop reçu.

TI est l'indicateur d'émission. Il est automatiquement mis à 1
à la transmission du 8eme bit en mode 0 ,
à la transmission du bit de stop dans les autres modes.
Avant une opération d'émission, il doit être remis à 0 par programme.

RI Indicateur de réception. Il est automatiquement mis à 1 , à la réception du 8eme bit en mode 0,
à la réception du bit de stop dans les autres modes (exception précisée dans la description de SM2).
RI doit être remis à 0 par programme.

Fonctionnement en registre à décalage (mode 0)

Les données série entrent et sortent par la broche RXD. La broche TXD représente alors l'horloge de décalage. Le format d'émission et de réception est de 8 bits (bit de poids faible en premier). La vitesse de communication est fixée à 1/12 de la fréquence de l'oscillateur du microcontrôleur La figure 11.2 présente un diagramme fonctionnel simplifié du port série en mode 0.

La transmission est initialisée par l'instruction qui utilise l'adresse SBUF comme adresse destination (exemple MOV SBUF, A). Le signal "Écriture dans SBUF" qui a lieu sur la dernière phase du cycle d'exécution de l'instruction (S6P2), provoque le chargement d'un "1" dans la neuvième position du registre à décalage de transmission et lance le cycle de transmission. Un cycle machine (douze périodes d'horloge) après l'apparition de ce même signal, la commande interne SEND est activée. SEND valide la sortie du registre à décalage sur la broche P3.0 (RXD) et valide aussi l'horloge de décalage sur P3.1 (TXD).

L'horloge de décalage est basse pendant S3, S4 et S5 de chaque cycle machine et à l'état haut pendant S6, S1 et S2. A l'instant S6P2 de chaque cycle machine, lorsque SEND est actif, le contenu du registre de transmission est décalé d'une position vers la droite. Lorsque les bits de données sortent à droite, des 0 sont introduits à gauche. Quand le bit le plus significatif de l'octet de donnée est en position de sortie, alors le 1 qui a été initialement chargé en neuvième position est juste à gauche du bit le plus significatif et toutes les autres positions sont occupées par des 0. Cette condition indique au bloc de contrôle TX l'ordre d'un dernier décalage suivi de la désactivation de SEND et de la mise à 1 de T1. L'ensemble de ces opérations a duré dix cycles machine après le signal d'écriture dans SBUF

La réception est initialisée par la condition REN=1 et RI=0. A la période S6P2 du prochain cycle machine, l'unité de contrôle RX inscrit les bits 11111110 dans le registre de réception et la phase d'horloge suivante active RECEIVE. RECEIVE active l'horloge de décalage sur la borne P3.1. L'horloge de décalage change d'état aux phases S3P1 et S6P1 de chaque cycle machine.

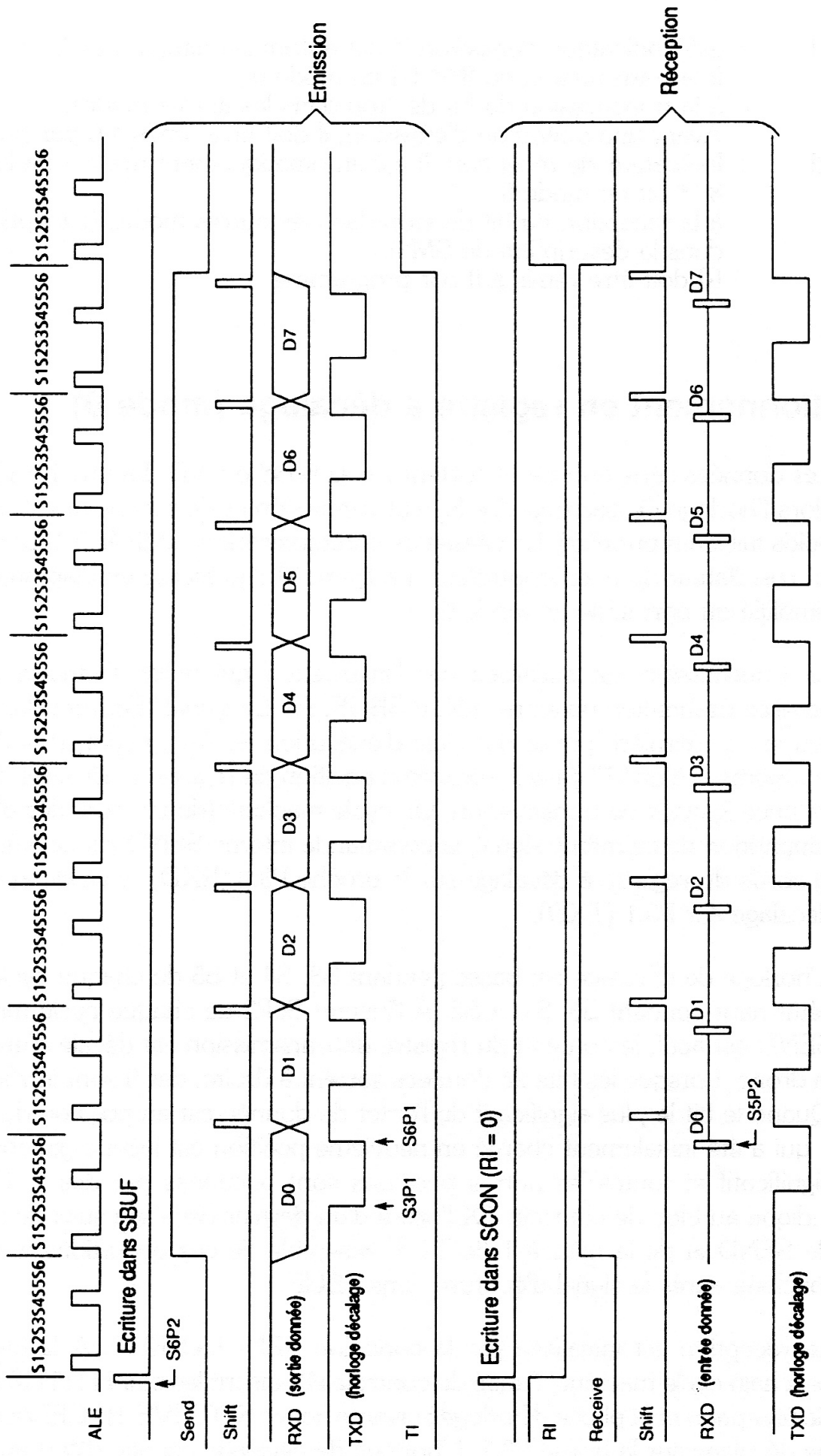


Figure 11.1 : Chronologie des opérations en mode 0 ***(d'après document SIEMENS)***

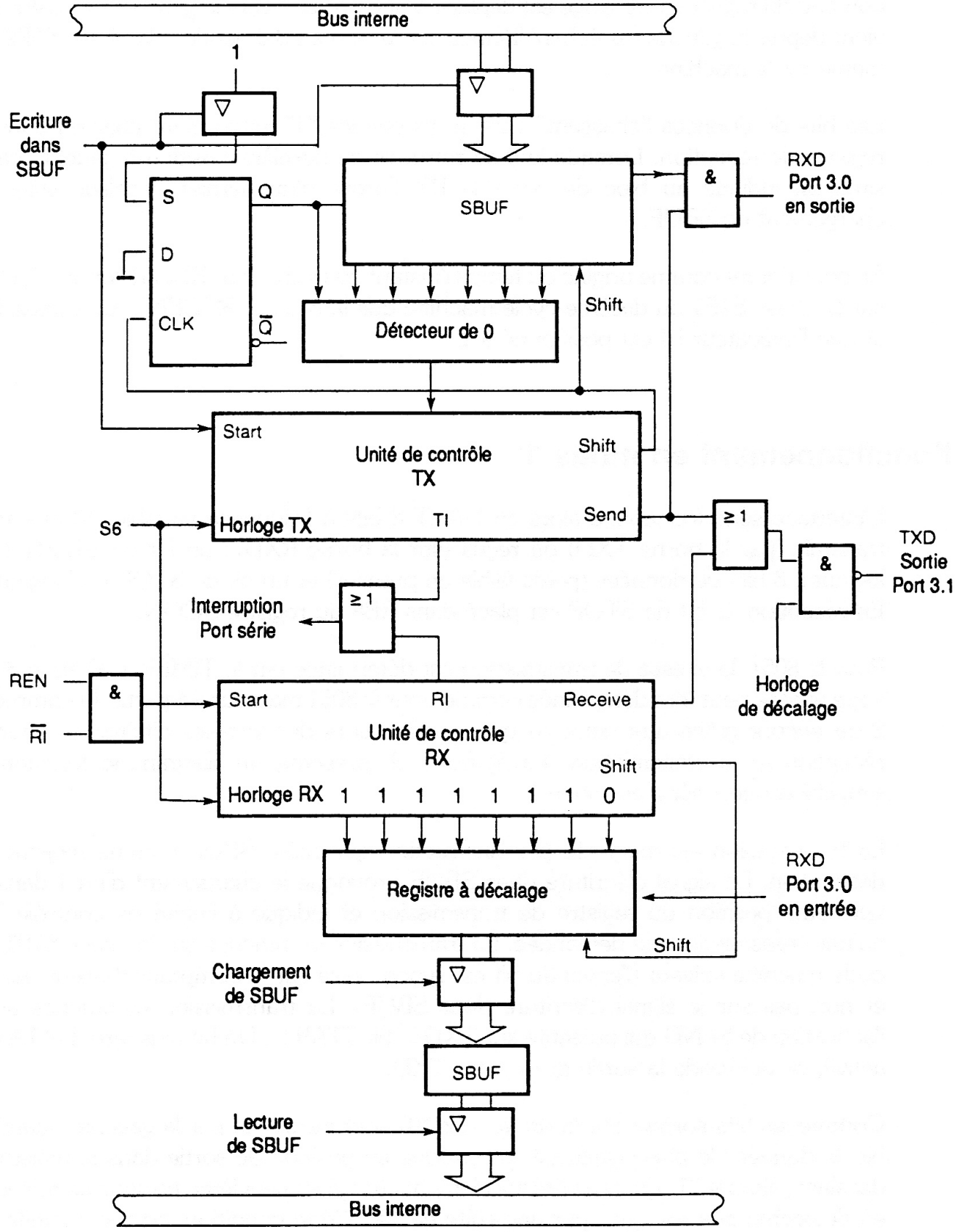


Figure 11.2 : Logigramme du port série en mode 0 ***(d'après document SIEMENS)***

Sur la phase S6P2 de chaque cycle machine pendant que RECEIVE est activé, le contenu du registre à décalage est déplacé d'une position vers la gauche. La valeur qui vient depuis la gauche est échantillonnée sur la borne P3.0 pendant la phase S5P2 du même cycle machine.

Les bits de données "chassent" vers la gauche les "1" initialement chargés dans le registre de réception. Lorsque le 0 se retrouve en dernière position à gauche, cette situation indique au bloc de contrôle RX l'ordre d'un dernier décalage suivi du chargement de SBUF

En considérant comme origine du temps l'instant où l'indicateur RI a été forcé à 0, c'est sur la phase S1P1 du dixième cycle machine que la bascule RECEIVE est remise à 0 et que l'indicateur RI est positionné à 1.

Fonctionnement en mode 1

L'interface série fonctionne alors en UART 8 bits à fréquence variable. 10 bits sont transmis (par la borne TXD) ou reçus (sur la borne RXD) · un bit de START ("0" logique), 8 bits de données (poids faible en premier) et un bit de STOP ("1" logique). En réception, le bit de STOP est placé dans RB8 du registre SCON.

Pour le 8051, la vitesse de transmission est déterminée par le TIMER 1. Pour le 8052 cette vitesse peut être déterminée comme pour le 8051 mais aussi à partir du compteur 2 ou encore grâce aux deux ce qui autorise alors des vitesses différentes pour la réception et la transmission. La figure 11.3. présente un diagramme fonctionnel simplifié du port série en mode 1.

La transmission est initialisée par l'instruction qui utilise SBUF comme registre de destination. Le signal d'écriture dans SBUF provoque le chargement d'un 1 dans la neuvième position du registre de transmission et indique à l'unité de contrôle TX qu'une transmission est demandée. La transmission commence sur la phase S1P1 du cycle machine suivant (l'envoi du bit est synchronisé sur le compteur diviseur par 16 et non pas sur le signal d'écriture dans SBUF). La transmission commence avec l'activation de SEND qui présente sur TXD le bit START Un bit plus tard, DATA est activé, ce qui valide la sortie du registre TXD.

Comme les bits sortent par la droite, des "0" sont introduits par la gauche, quand le bit de données le plus significatif se retrouve en position de sortie dans le registre à décalage, alors le "1" qui avait été initialement chargé en neuvième position se retrouve en deuxième position suivi par une suite de "0" Cette condition indique à l'unité de contrôle TX l'ordre d'un dernier décalage, la désactivation de SEND et la mise à 1 de TI.

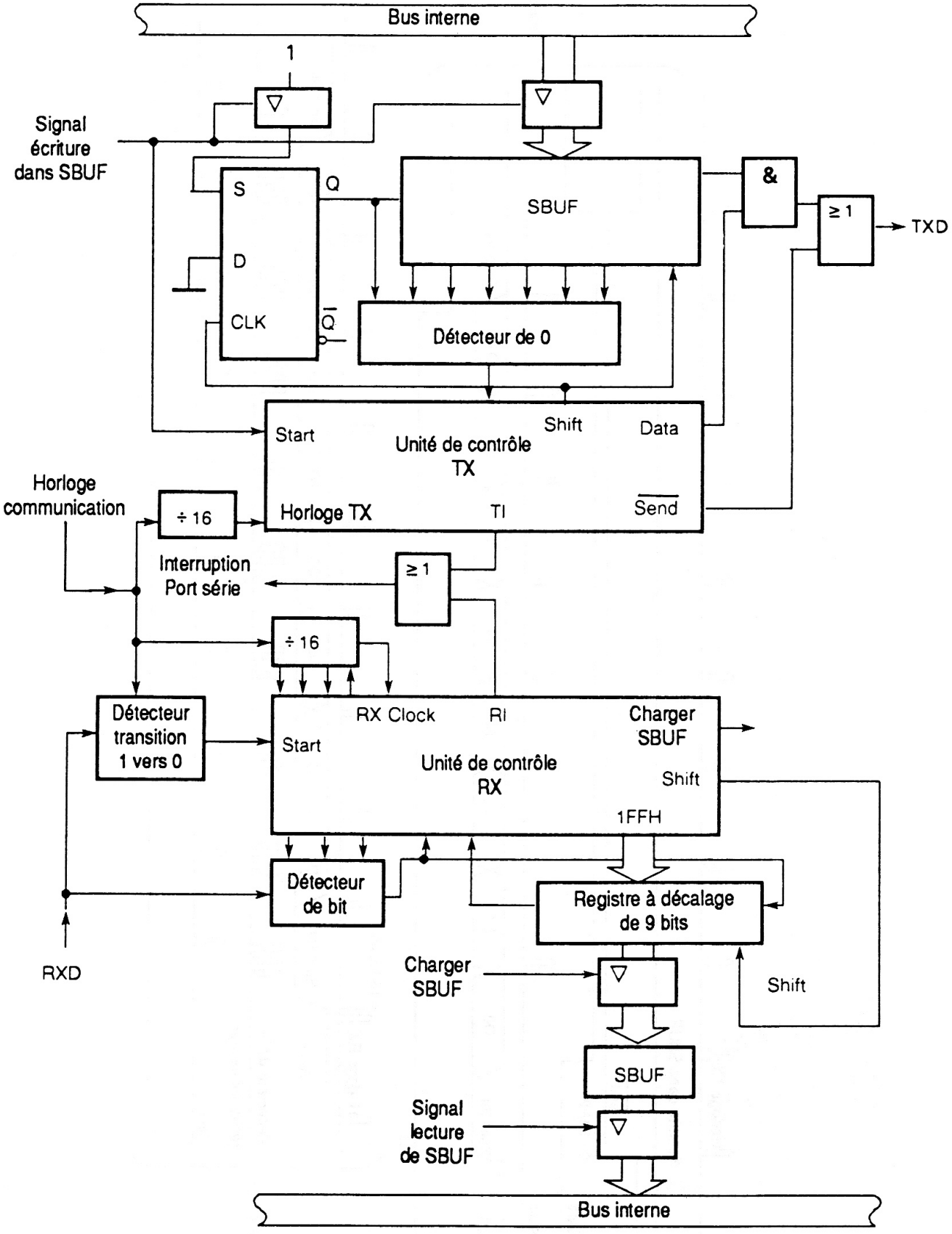


Figure 11.3 : Port série en mode 1 ***(d'après document SIEMENS)***

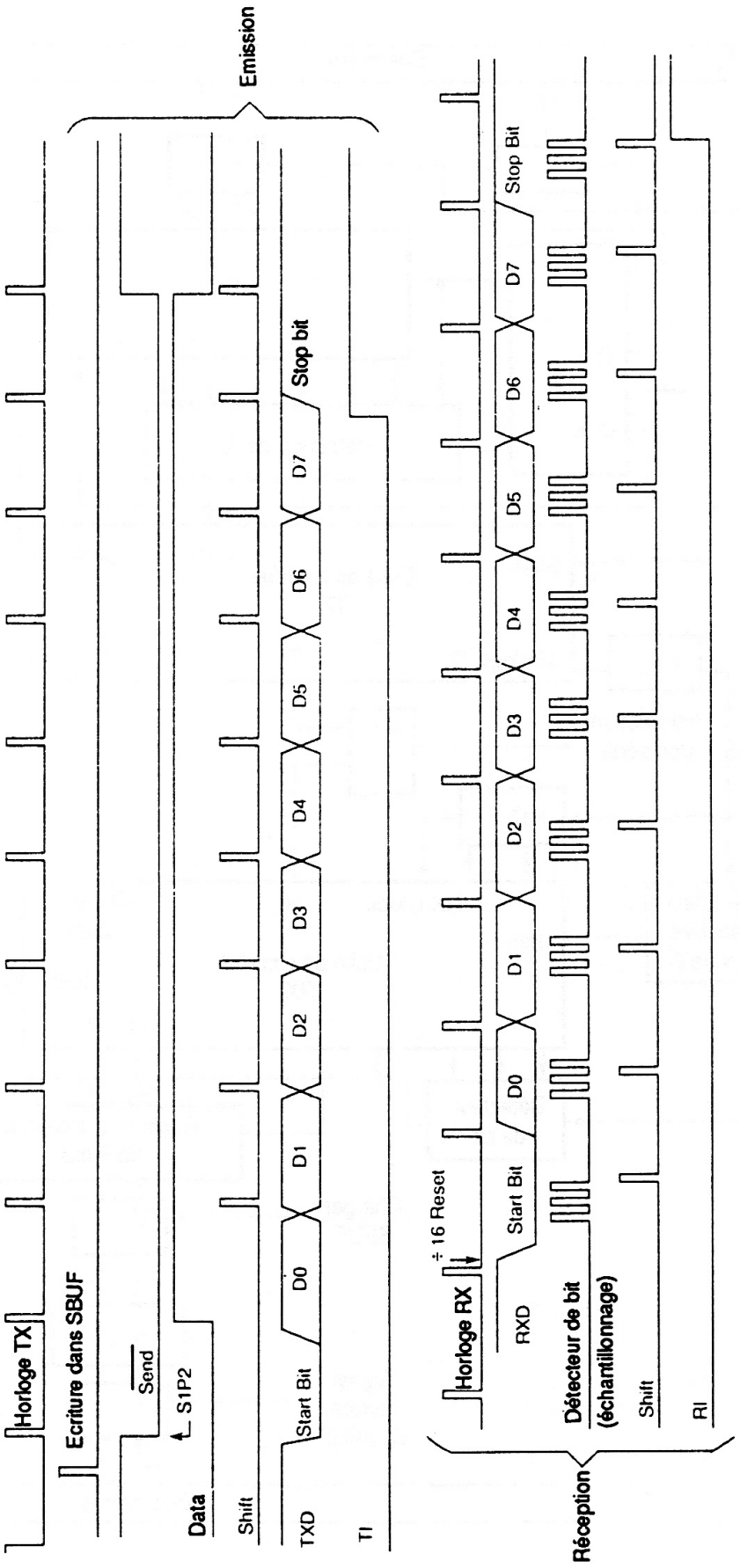


Figure 11.4 : Chronologie des opérations en mode 1 ***(d'après document SIEMENS)***

La réception est initialisée par la détection d'une transition de 1 vers 0 sur la broche RXD. Pour cette raison, RXD est échantillonné à une fréquence de 16 fois supérieure à la fréquence de réception. Dès la détection de cette transition le compteur par 16 est immédiatement initialisé et la valeur 1FFH est écrite dans le registre à décalage.

Les seize périodes du compteur par 16 permettent une division en seizièmes de la période de réception d'un bit. Aux septième, huitième et neuvième périodes de ce compteur le détecteur de bit échantillonne la valeur de RXD. La valeur retenue est le valeur "vue" au moins deux fois sur les trois. Cette méthode améliore la réjection du bruit. Si le premier bit détecté n'était pas un "0" (bit de START) le circuit de réception serait réinitialisé dans l'attente d'une nouvelle transition de 1 vers 0.

Dans le cas où le bit de START est valide, celui-ci est décalé vers l'entrée du registre à décalage et la réception des bits suivants est possible. Comme les bits arrivent depuis la droite, la sortie des 1 se fait par la gauche. Quand le bit de START se retrouve en position de sortie (dans le registre 9 bits pour le mode 1), l'unité de contrôle RX exécute un dernier décalage, charge la donnée dans SBUF et RB8 et positionne à 1 l'indicateur RI. Le signal de chargement de SBUF, deRB8 et le positionnement de RI n'ont lieu que si, et seulement si, la condition suivante est présente durant le dernier décalage

RI = 0 et (SM2 = 0 ou STOP bit = 1)

Si cette condition n'est pas vraie la trame de réception est perdue et RI n'est pas positionné.

Si la condition est vraie, le bit de STOP est placé dans RB8 et les 8 bits de données sont chargés dans SBUF A cet instant, que la condition ait été remplie ou pas, l'unité reprend l'attente d'une transition 1 vers 0 sur RXD.

Fonctionnement en mode 2 et 3

L'interface série fonctionne en UART dont le format des données est de 9 bits.

Onze bits sont transmis ou reçus (par les bornes TXD ou RXD). Un bit de départ (start bit = 0), 8 bits de données, un neuvième bit programmable, et un bit de stop (stop bit = 1). Pour la transmission, le neuvième bit (programmé dans TB8 du registre SCON) peut prendre les valeurs 0 ou 1, ou bien, par exemple servir de bit de parité si on lui affecte la valeur de l'indicateur P de PSW par l'intermédiaire de TB8. En réception ce neuvième bit sera placé dans RB8 de SCON.

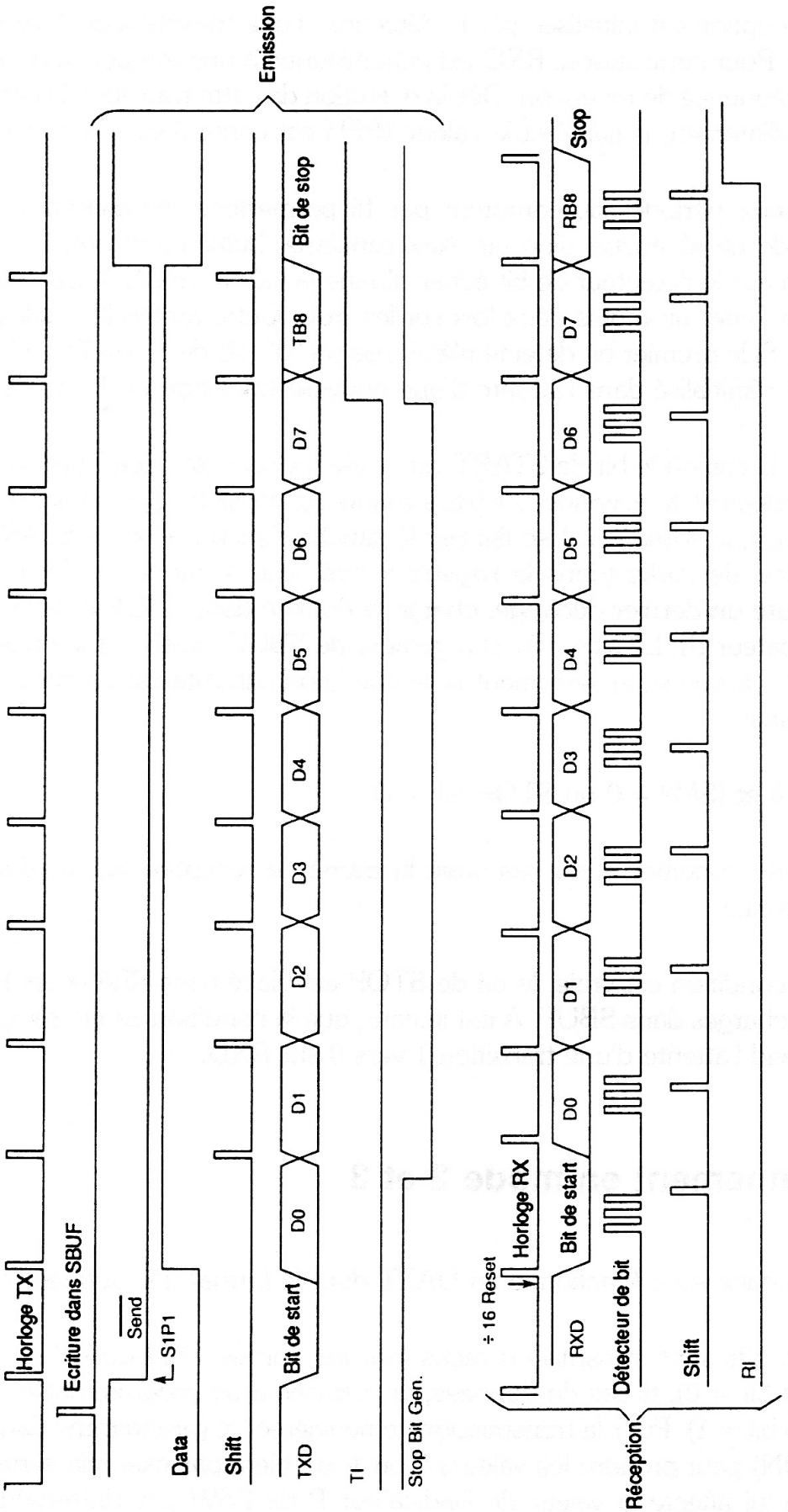


Figure 11.5 : Chronologie des opérations en mode 2 et 3 *(d'après document SIEMENS).*

Pour le mode 2, la vitesse d'émission et de réception est programmable soit 1/32 ou 1/64 de la fréquence oscillateur du microcontrôleur Le choix d'une des deux fréquences est déterminé par l'état de SMOD, bit 7 du registre PCON (Power Control Register). Ce bit est automatiquement initialisé à l'état 0 et permet d'obtenir une fréquence de communication 64 fois plus petite que celle de l'oscillateur du microcontrôleur. Lorsque l'utilisateur positionne ce bit à 1, la fréquence de communication est doublée.

Pour le mode 3, la vitesse d'émission et de réception est obtenue à partir du compteur/temporisateur 1 (1 ou 2 pour le 8052). Les figures 11.6. et 11.7 donnent les diagrammes fonctionnels du port série dans ces deux modes.

La transmission est initialisée par chaque instruction qui utilise SBUF comme registre de destination. Le signal d'écriture vers SBUF charge dans la neuvième position du registre à décalage, le neuvième bit de la donnée situé dans TB8 et indique à l'unité de contrôle TX qu'une transmission est demandée. La transmission commence à la période S1P1 du cycle machine suivant le passage à 0 du compteur diviseur par 16.

La transmission commence avec l'activation de SEND qui place le bit de START sur TXD. Un bit plus tard, DATA est activé, ce qui valide le bit de sortie du registre à décalage pour TXD. La première impulsion de décalage apparaît un bit plus tard.

Cette première impulsion introduit le 1 du bit STOP en neuvième position du registre à décalage. Puis c'est une suite de 0 qui est introduite depuis la gauche lors des décalages. Lorsque le bit de TB8 est en position de sortie, le bit de stop juste à sa gauche, tous les autres bits du registre à décalage sont à 0. Cette condition indique à l'unité de contrôle TX l'ordre d'un dernier décalage, de désactiver SEND et de positionner à 1 l'indicateur TI. Cette situation se produit à la douzième révolution du compteur diviseur par 16 après écriture dans SBUF

La réception est initialisée par la détection d'une transition 1 vers 0 sur la broche RXD. Pour ce faire, la broche RXD est échantillonnée à une fréquence 16 fois supérieure à la fréquence de réception qui a été établie. Lorsque la transition a été détectée, le compteur diviseur par 16 est réinitialisé et la valeur 1FFH est écrite dans le registre à décalage d'entrée.

Les seize périodes du compteur par 16 permettent une division en seizièmes de la période de réception d'un bit. Aux septième, huitième et neuvième périodes de ce compteur le détecteur de bit échantillonne la valeur de RXD. La valeur retenue est le valeur "vue" au moins deux fois sur les trois. Cette méthode améliore la réjection du bruit. Si le premier bit détecté n'était pas un "0" (bit de START) le circuit de réception serait réinitialisé dans l'attente d'une nouvelle transition de 1 vers 0. Dans le cas où le bit de START est valide celui-ci est décalé vers l'entrée du registre à décalage et la réception des bits suivants est possible.

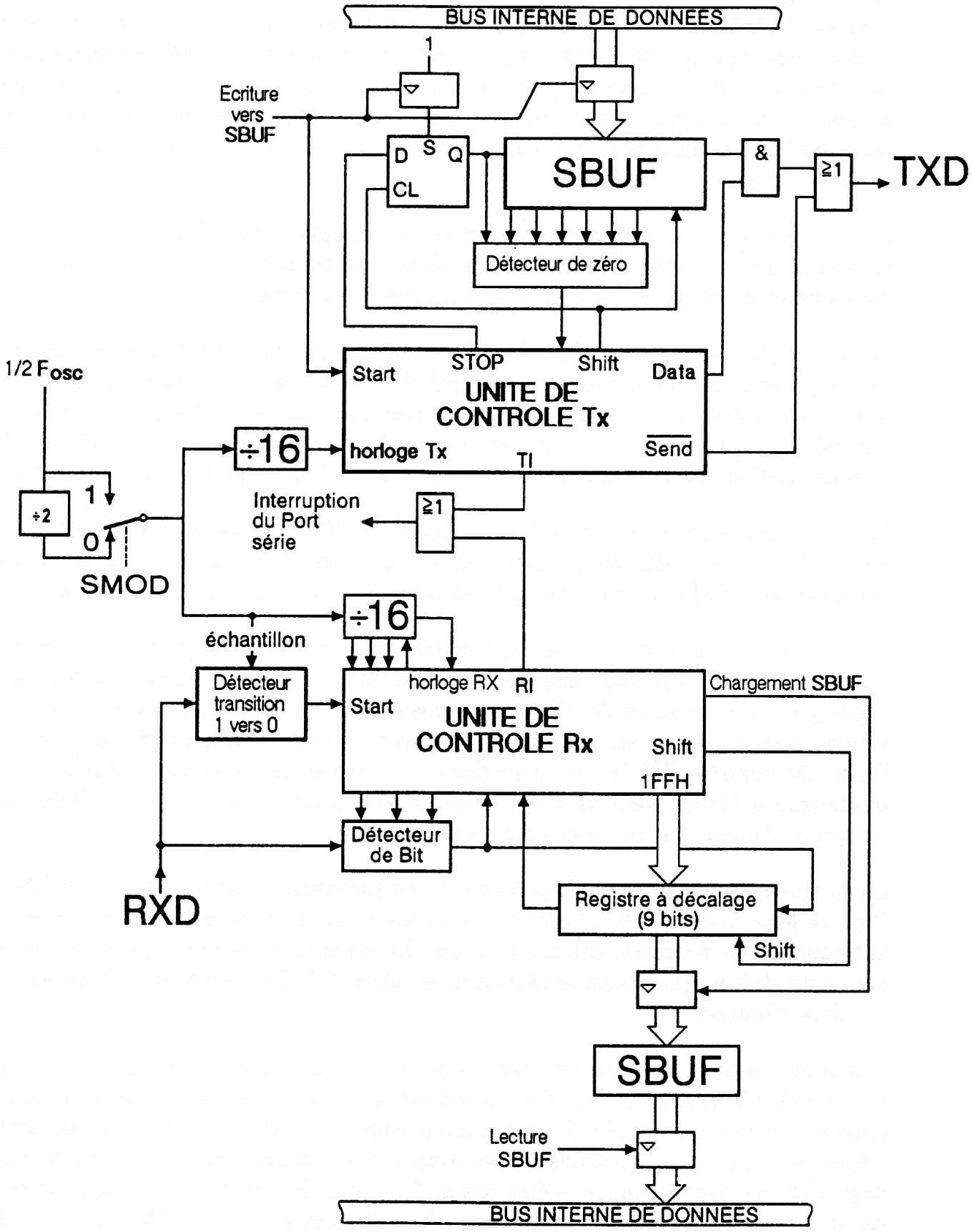


Figure 11.6 : Port série en mode 2 ***(d'après document SIEMENS).***

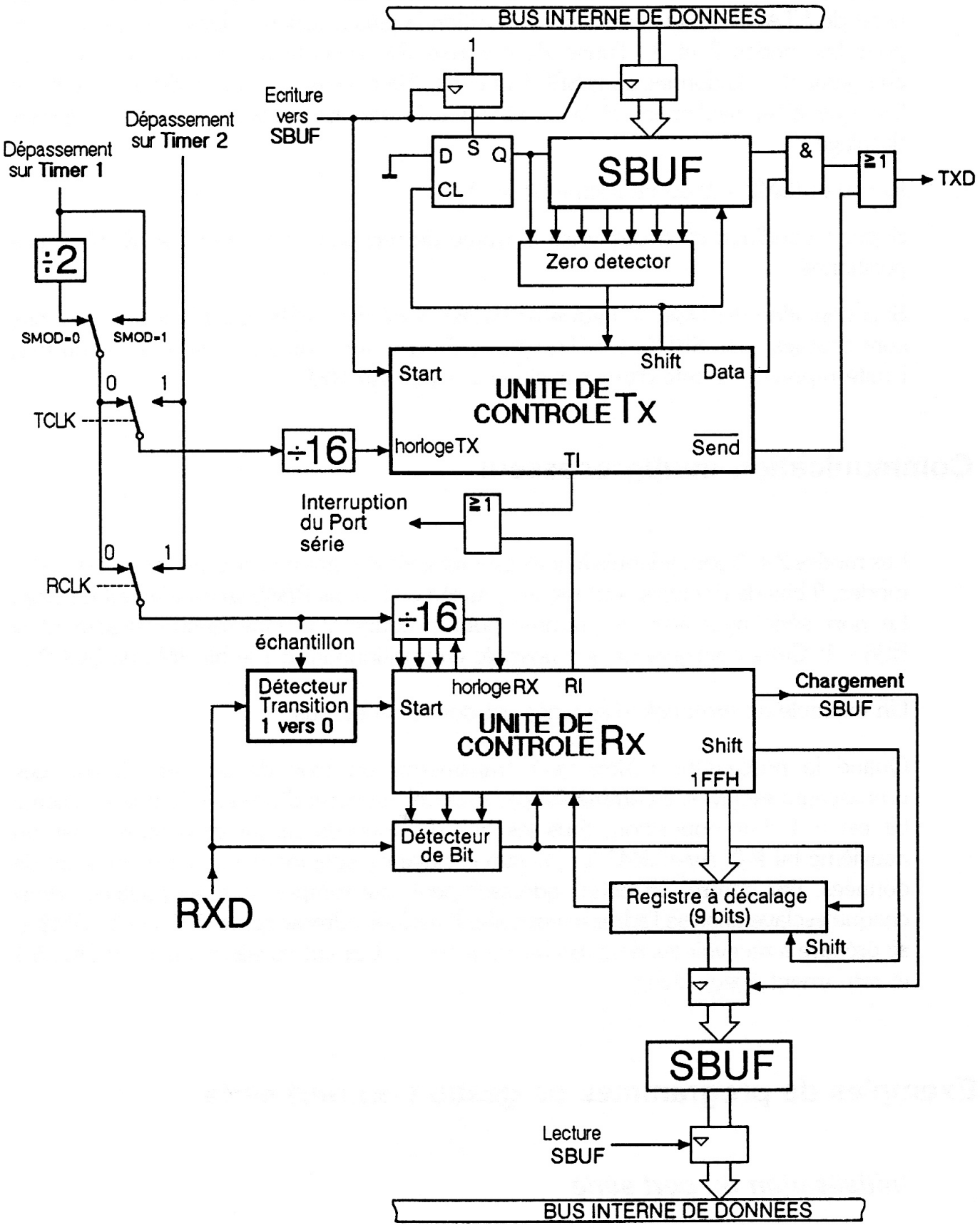


Figure 11.7 : Port série en mode 3 ***(d'après document SIEMENS)***

Comme les bits arrivent depuis la droite, la sortie des 1 se fait par la gauche. Quand le bit de START se retrouve dans la position la plus à gauche (dans le registre 9 bits pour les modes 2 et 3), l'unité de contrôle RX exécute un dernier décalage. Le chargement de la donnée dans SBUF et dans RB8 et la mise à 1 de l'indicateur RI n'ont lieu que si et seulement si la condition suivante est présente durant le dernier décalage

RI = 0 et (SM2 = 0 ou neuvième bit = 1)

Si cette condition n'est pas vraie la trame de réception est perdue et RI n'est pas positionné.

Si la condition est vraie, le neuvième bit est placé dans RB8 et les huit premiers bits sont chargés dans SBUF Un "bit plus tard", que la condition ait été remplie ou pas, l'unité reprend l'attente d'une transition 1 vers 0 sur RXD.

Communication multiprocesseur

Les modes 2 et 3 sont adaptés à une communication entre microcontrôleurs. Dans ces modes, 9 bits de données sont reçus (neuvième bit dans RB8), suivis d'un bit de stop. Le port série peut être programmé pour produire une interruption seulement si RB8 = 1. Cette configuration est possible en positionnant à 1 le bit SM2 de SCON.

Un exemple de protocole d'échange est donné ci-dessous

Quand le processeur maître veut transmettre un bloc de données à l'un des processeurs esclaves, il commence par envoyer un octet d'adresse dont le neuvième bit est à 1. Par opposition, tous les bits de données seront caractérisés par un neuvième bit à 0. Avec SM2 à 1, aucun esclave ne sera interrompu par un octet de données. Par contre, un octet "adresse" peut interrompre tous les esclaves. Ainsi chaque esclave analyse l'adresse envoyée. L'esclave adressé remet à 0 son bit SM2 et se prépare à recevoir les données qui vont suivre. Les autres laissent leur bit SM2 à 1 et retournent à leur tâche.

Exemples de programmes de gestion du port série

Initialisation du port série

Soit à réaliser une communication série au format 8 bits à une vitesse de 4 800 bauds pour un microcontrôleur 8031 dont la fréquence du quartz est de 11 059 MHz.

D'après les indications données dans le paragraphe 11.2 il faut choisir le mode 1. Dans ce mode, le bit SM2 peut être mis à 0 ou à 1. Si ce bit est mis à 1, l'indicateur de réception ne sera positionné à 1 que pour une réception se terminant par un bit de STOP valide. La réception est autorisée par la mise à 1 de REN. Les autres bits du registre SCON sont initialisés comme l'indique le tableau récapitulatif suivant

SCON 98H

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
0	1	1/0	1	1	0	1	0

Bien que non utilisé dans le mode 1, le bit TB8 peut être mis à 1. Le bit RB8 mis à 0, sera modifié lors d'une réception. TI est mis à 1 pour indiquer que le tampon d'émission est disponible.

La fréquence de communication de 4800 bauds sera obtenue à partir du timer 1. Par initialisation du registre TMOD on définit un mode de rechargement automatique sur le Timer 1. Le registre TH1 du compteur est chargé de la valeur FAH (voir tableau 10.11 de la page p. 127). Cette valeur sera rechargée automatiquement dans TL1 mais il n'est pas inutile d'initialiser TL1 afin que la fréquence d'horloge soit stable dès le lancement du Timer

TMOD

TIMER 1				TIMER 0			
GATE	C/T	M1	M0	GATE	C/T	M1	M0
0	0	1	0	X	X	X	X

GATE = 0 GATE inactive

C/T = 0 le TIMER est en fonction **temporisateur**

M1 = 1 ET M0 = 0 compteur 8 bits à rechargement automatique. Le contenu du registre TH1 est rechargé dans le registre TL1 lorsque la valeur de celui-ci repasse à 0 , d'où le programme d'initialisation

```
MOV   SCON,#01011010B     ,
MOV   TMOD,#00100000B     , 8 BIT AUTO RELOAD TIMER 1
MOV   TH1,#0FAH           , VALEUR POUR 4800 BAUDS
MOV   TL1,#0FAH           ,
SETB  TR1                 , ACTIVE L'HORLOGE
```

Réception et émission

La réception d'un code valide est indiquée par la mise à 1 de l'indicateur RI. Une émission est possible lorsque le tampon de sortie est vide (TI=1)

Les sous-programmes suivants restent simples mais n'utilisent pas les avantages d'une gestion sous interruption.

```
EMISSION    JNB  TI,EMISSION    , ATTENDRE QUE TI=1 : TAMPON VIDE
            MOV  SBUF,A         , PUIS ENVOI DU CODE DANS A
            CLR  TI             , REMISE INDICATEUR A 0
            RET
RECEPTION   JNB  RI,RECEPTION   ; ATTENDRE RECEPTION COMPLETE
            MOV  A,SBUF         , PUIS PLACER CODE DANS ACCUMULATEUR
            CLR  RI             , REMETTRE INDICATEUR A 0
            RET                 ET RETOUR
```

12

Les interruptions

Le noyau 8051 possède cinq sources d'interruption, alors que la version 8052 en comporte six comme l'indique la figure 12.1. Toutes ces sources d'interruption sont du type "masquable", c'est-à-dire autorisables ou non par logiciel. Après initialisation toutes les interruptions sont inhibées.

Le registre IE est dédié à l'autorisation ou l'inhibition individuelle des sources d'interruption.

Le mécanisme d'interruption est vectorisé sur adresse fixe. Ces adresses sont situées dans la partie basse de la mémoire programme.

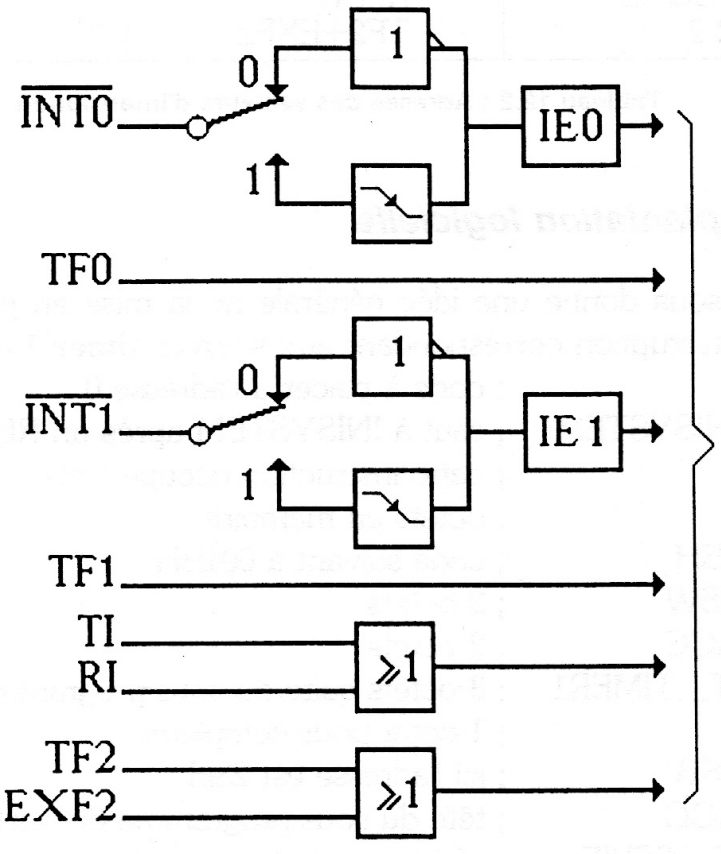


Figure 12.1 : Sources d'interruption (TF2 et EXF2 n'existent que dans les versions 8052)

Table des vecteurs

La logique d'implantation des vecteurs d'interruption est la suivante :

à l'adresse 0000 de la mémoire programme correspond la première instruction à exécuter après une initialisation (RESET = 1). Le constructeur a réservé 3 octets pour cette première instruction, valeur correspondant à une instruction de saut du type "**LJMP adr**".

C'est donc à l'adresse 0003 que se situe le premier emplacement réservé aux sous-programmes devant gérer les différentes sources d'interruption du programme principal. Chaque adresse correspondant à l'un de ces vecteurs est espacée de huit octets. Cet espace est dans la plupart des cas simplement suffisant à recevoir la "tête" du sous-programme de service de l'interruption. La table suivante précise les sources d'interruption, l'indicateur correspondant et l'adresse du vecteur propre à cette source :

Source d'interruption	Indicateur	Adresse du vecteur
INT0	IE0	0003H
TIMER 0	TF0	000BH
INT1	IE1	0013H
TIMER 1	TF1	001BH
PORT SERIE	RI+TI	0023H
TIMER 2	TF2+EXF2	002BH

Tableau 12.2 : Adresse des vecteurs d'interruption

Exemple d'implantation logicielle

L'exemple ci-dessous donne une idée générale de la mise en place de deux sous-programmes d'interruption correspondant aux sources **timer** 1 et **port serie** :

```
ORG     0           ; code à placer à l'adresse 0
LJMP    INISYSTEM   ; saut à INISYSTEM après un RESET
                    ; cette instruction occupe trois
                    ; octets en mémoire.
ORG     1BH         ; code suivant à 001BH
PUSH    PSW         ; 2 octets
PUSH    ACC         ; 2 octets
LJMP    IT_TIMER1   ; 3 octets suite du sous-programme
NOP                 ; 1 octet pour compléter
PUSH    PSW         ; ici l'adresse est 23H
PUSH    ACC         ; tête du sous-programme du port
LJMP    IT_SERIE    ; série puis saut vers suite...
NOP                 ; un NOP si int. TIMER utilisée
```

```
        .....
IT_TIMER1 :
        .....                   ; ici suite d'instructions du sous-
        .....                   ; programme pour gestion TIMER1
        .....                   ; cette suite se termine par...
        POP     ACC
        POP     PSW
        RETI
IT_SERIE :
        .....                   ; idem pour le port série
        .....
        POP     ACC
        POP     PSW
        RETI
INISYSTEM :
        .....                   ; initialisation et programme princ.
```

Exécution d'une interruption

Lorsque le processeur accepte une requête d'interruption, un saut automatique au sous-programme de service correspondant à la source est exécuté. Pour plus de précision le saut est comparable à un appel de sous-programme, c'est-à-dire qu'avant d'exécuter le branchement au sous-programme, le processeur sauvegarde automatiquement sur la pile l'adresse de la prochaine instruction qui devait normalement être exécutée. Un indicateur interne est positionné pour signaler le déroulement d'une interruption, puis le saut est exécuté. Un sous-programme de service d'interruption doit obligatoirement se terminer par l'instruction particulière de retour RETI. Elle permet tout comme l'instruction RET de récupérer depuis la pile l'adresse à laquelle le programme interrompu doit être repris, mais aussi de mettre à jour l'indicateur interne autorisant ainsi la prise en compte d'une autre interruption de même niveau.

Autorisation et priorité des interruptions

A l'initialisation du microcontrôleur, toutes les sources d'interruption sont inhibées par la remise à zéro des bits du registre IE. Afin d'utiliser le mécanisme des interruptions, il est donc nécessaire de commencer par autoriser la ou les sources d'interruption utilisées par l'application. La fonction des 8 bits du registre IE est donnée dans le tableau 12.3.

Ce registre fait partie de la RAM interne adressable au niveau du bit, il est donc possible par une seule instruction, de modifier un seul de ces bits sans modifier l'état des autres. Cette précision justifie le rôle du bit EA. En effet, quel que soit l'état des

7 autres bits, il est possible de suspendre la prise en compte de toute interruption en positionnant EA à 0, puis la remise à 1 de ce même bit autorise à nouveau la prise en compte des interruptions définies par l'état des 7 autres bits.

IE : Adresse directe A8H et adressable au niveau du bit

AF	AE	AD	AC	AB	AA	A9	A8
EA	—	ET2	ES	ET1	EX1	ET0	EX0

position	symbole	fonction
IE.7	EA	Inhibition de toutes les interruptions
IE.6	—	sans fonction pour 8051 et 8052
IE.5	ET2	interruption TIMER 2
IE.4	ES	interruption PORT SERIE
IE.3	ET1	interruption TIMER 1
IE.2	EX1	interruption EXTERNE depuis broche INT1
IE.1	ET0	interruption TIMER 0
IE.0	EX0	interruption EXTERNE depuis broche INT0

Tableau 12.3 : Registre d'autorisation des interruptions IE

Exemple : Dans un programme on désire prendre en compte les interruptions provenant du port série et de la broche externe INT0.

L'instruction MOV IE,#00010001B

prépare la prise en compte des deux sources choisies, puis

SETB EA

autorise la prise en compte des interruptions alors que

CLR EA

suspend cette prise en compte sans modifier IE.0 et IE.4

Il est recommandé de laisser à 0 les bits inutilisés (bit IE.6 et IE.5 pour le 8051) afin de conserver au programme une compatibilité avec des microcontrôleurs de version 8052 ou autres.

Dans l'exemple cité ci-dessus, deux sources d'interruption sont prises en compte par le microcontrôleur. Il est important de savoir laquelle de ces deux sources serait prise en compte dans le cas où sur le même cycle machine, les deux sources seraient échantillonnées comme active.

Il faut savoir que pour l'instant la mise en place des autorisations des interruptions par le positionnement des bits du registre IE confère un ordre de prise en compte intrinsèque au processeur même si toutes les sources sont considérées d'un même

niveau de priorité. En effet l'échantillonnage des demandes est fait selon une séquence de test définissant une hiérarchie implicite des prises en compte des requêtes d'interruption simultanées. Cette hiérarchie correspond à l'ordre de la table des vecteurs et elle est redonnée ci-dessous :

ordre	source	
1	IE0	plus haute priorité naturelle
2	TF0	
3	IE1	
4	TF1	
5	RI+TI	
6	TF2+EXF2	plus basse priorité

Dans notre exemple où INT0 et le port série émettraient en même temps une demande d'interruption, c'est la source INT0 (indicateur IE0) qui serait servie en premier.

Modification de l'ordre de priorité

Il est possible de modifier le niveau implicite de priorité des six sources d'interruption en attribuant à une ou plusieurs de ces sources un niveau supérieur de prise en compte. Cette modification est réalisable par programmation des bits du registre IP (*Interrupt Priority*). Chacun des bits de ce registre, lorsqu'il est positionné à 1, donne à la source correspondante un niveau de priorité supérieur aux demandes émanant des sources de niveau initial. La figure 12.4 permet de situer le rôle des deux registres de contrôle IE et IP.

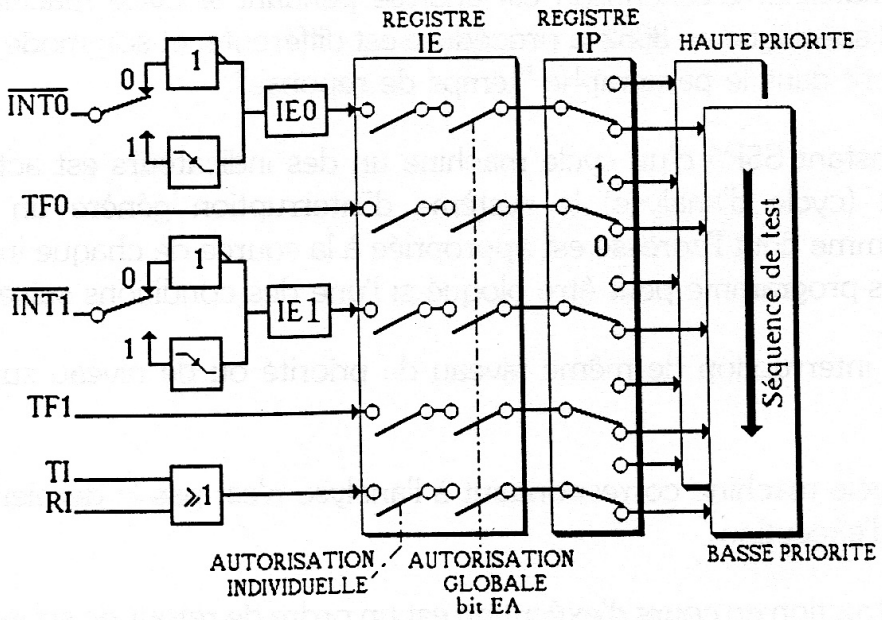


Figure 12.4 : Structure générale du système d'interruption (noyau 8051)

Dans le cas où deux indicateurs d'interruption affectés d'un niveau de priorité supérieur sont actifs en même temps, c'est alors la hiérarchie implicite déterminée par la séquence de test qui détermine l'ordre de prise en compte de ces deux interruptions.

Le tableau 12.5 donne le détail d'affectation du registre IP.

IP : Adresse directe B8H et adressable au niveau du bit

BF	BE	BD	BC	BB	BA	B9	B8
—	—	PT2	PS	PT1	PX1	PT0	PX0

position	symbole	fonction
IP.7	—	réservé à une fonction future
IP.6	—	réservé à une fonction future
IP.5	PT2	priorité interruption TIMER 2 (sur 8052)
IP.4	PS	priorité interruption port série
IP.3	PT1	priorité interruption TIMER 1
IP.2	PX1	priorité interruption externe INT1
IP.1	PT0	priorité interruption TIMER 0
IP.0	PX0	priorité interruption externe INT0

Tableau 12.5 : Registre de priorité des interruptions IP

Prise en compte d'une demande d'interruption

Les indicateurs d'interruption sont échantillonnés durant la période S5P2 de chaque cycle machine. L'échantillon est analysé pendant le cycle machine suivant. Pour le timer 2 spécifique au 8052 la procédure est différente, et son mode de prise en compte est décrit dans le paragraphe "temps de reponse".

Si à l'instant S5P5 d'un cycle machine un des indicateurs est actif, à la fin du cycle suivant (cycle d'analyse) le système d'interruption génère un appel à un sous-programme dont l'adresse est appropriée à la source de chaque indicateur. Cet appel de sous-programme peut être bloqué si l'une des conditions suivantes est détectée :

1 - une interruption de même niveau de priorité ou de niveau supérieur est déjà en cours

2 - le cycle machine correspondant à l'analyse n'est pas le dernier de l'instruction en cours d'exécution

3 - l'instruction en cours d'exécution est un ordre de retour de sous-programme (RETI) ou une instruction correspondant à une écriture dans l'un des registres IE ou IP.

La condition 3 assure l'exécution d'une instruction après modification des registres définissant le mécanisme des interruptions et après le saut correspondant au retour de traitement de l'interruption. C'est seulement après l'exécution de cette instruction supplémentaire qu'une autre requête d'interruption sera prise en compte et traitée.

Dans le cas d'un blocage du processus d'interruption par l'une des trois conditions citées qu'advient-il de la requête non servie ?

Etant donné que l'échantillonnage des indicateurs d'interruption est renouvelé à chaque cycle machine, le nouvel échantillon remplace donc l'ancien et si la demande d'interruption est toujours présente, elle sera prise en compte sur le cycle machine suivant. Dans le cas où la demande d'interruption est issue d'une source interne (port série, timer...) l'indicateur restera positionné car celui-ci est soit inchangé, par principe, soit remis à zéro au moment du saut au sous-programme de service. Donc la demande d'interruption sera prise en compte même après blocage, sauf dans le cas où, par logiciel, l'utilisateur remet de lui même l'indicateur à 0 !

Par contre dans le cas d'une interruption de source externe programmée sur niveau logique, il est possible qu'une demande d'interruption ne soit jamais prise en compte. En effet, sur blocage de demande il faut être sûr que le changement d'état persistera jusqu'au prochain échantillonnage, soit un cycle machine plus tard, pour qu'une nouvelle analyse conduise à l'exécution de la demande.

Lorsque le processeur accepte la requête d'interruption, le saut automatique au sous-programme approprié est exécuté. C'est à ce moment-là qu'éventuellement l'indicateur de demande d'interruption est remis à zéro. Pour plus de précision il est nécessaire d'analyser en détail les particularités de chaque source d'interruption.

Interruptions extérieures INT0 ET INT1

Deux sources d'interruption extérieure sont possibles à partir de deux lignes du port P3. Il s'agit des lignes P3.2 et P3.3 dont les fonctions secondaires sont respectivement INT0 et INT1. Une interruption extérieure peut être définie comme survenant pour un changement d'état logique sur l'une de ces deux lignes (interruption sur niveau) ou sur détection d'un changement d'état (interruption sur front négatif ou descendant), le choix est réalisé par programmation des deux bits IT0 et IT1 situés dans le registre TCON (voir chapitre timer).

Si le bit IT0 (ou IT1) est mis à 1, l'interruption surviendra pour un changement d'état 0 vers 1 (front descendant).

Si le bit IT0 (ou IT1) est mis à 0, l'interruption surviendra pour la présence d'un état logique 0 sur la broche INT0 (ou INT1).

Dans le cas d'une interruption par front la broche d'entrée est échantillonnée pendant la durée d'un cycle machine. Pour que la transition soit correctement prise en compte le niveau 1 et le niveau 0 encadrant la transition doivent donc durer au moins un temps égal à un cycle machine, c'est-à-dire un temps égal à 12 périodes d'horloge (1 microseconde pour un quartz de 12 MHZ).

L'état de ces sources d'interruption est signalé par deux autres bits de ce même registre, il s'agit des bits IE0 et IE1. Ces bits sont mis à 1 lorsque la condition d'interruption est remplie (transition ou changement d'état), alors si et seulement si la source d'interruption est autorisée, le compteur programme est positionné à l'adresse du sous-programme d'interruption de INT0 ou INT1.

Après branchement à cette adresse, le bit IE0 (ou IE1) est remis à 0 automatiquement lorsque l'événement à détecter est un front descendant.

Par contre, si l'événement à détecter est un changement d'état logique, l'indicateur IE0 (ou IE1) reste à 1 et doit être remis à 0 par programmation.

Interruption par les timers

Sources d'interruption timer 0 et 1

Pour ces deux sources, les demandes d'interruption sont signalées par les indicateurs TF0 et TF1 correspondant respectivement aux bits 7 et 5 du registre TCON. Ces indicateurs sont mis à 1 lors d'un débordement de capacité du compteur (exception faite pour le timer 0 fonctionnant en mode 3). Lorsqu'une "*interruption timer*" est honorée l'indicateur (TF0 ou TF1) est remis automatiquement à 0 au moment du saut vers le sous-programme de service (adresse 000BH pour TF0 et 001BH pour TF1)

Timer 2 du 8052

Une interruption du timer 2 est obtenue à partir des deux indicateurs TF2 et EXF2. La prise en compte est faite sur le résultat d'une opération logique OU entre ces deux indicateurs. Il suffit donc qu'un des deux indicateurs passe à l'état 1 pour qu'une demande d'interruption soit reconnue. L'état de ces indicateurs n'est pas modifié au moment du saut à l'adresse 002BH. Cette particularité permet, par programme, d'analyser l'état de chaque indicateur afin d'envisager éventuellement deux traitements différents. Le sous-programme de service devra comporter la ou les instructions de remise à 0 du ou des indicateurs avant retour au programme principal.

Port série

Comme pour le timer 2, les demandes d'interruptions à partir du port série sont signalées par deux indicateurs. Il s'agit de RI qui indique une demande d'interruption sur réception d'un octet et de TI qui signale que le tampon de transmission est libre à l'envoi d'un nouvel octet. Au moment du saut vers l'adresse 0023H ces indicateurs ne sont pas automatiquement remis à 0. Cette opération est laissée au soin de l'utilisateur après analyse de la nature même de l'interruption : réception d'une donnée ou transmission possible. La demande d'interruption est reconnue sur opération logique OU entre les deux indicateurs RI et TI.

Temps de réponse

Du fait qu'une demande d'interruption puisse être bloquée sur condition particulière et vu la nature différente des sources d'interruption, il est évident que le temps de réponse, entre le moment où le phénomène génère la demande d'interruption et le moment où le sous-programme de service est exécuté, ne soit pas constant. Il reste à définir les temps de réponse minimal et maximal pour tirer profit des possibilités du microcontrôleur. Pour cela il est nécessaire de reprendre tous les cas de figure résultant des particularités du mécanisme de service d'une interruption.

En ce qui concerne les sources externes, l'état présent sur les broches INT0 et INT1 est inversé et mémorisé dans les indicateurs IE0 et IE1 durant la période S5P2 de chaque cycle machine.

De façon similaire, les indicateurs correspondant aux sources internes

— EXF2 du timer 2,

— RI et TI du port série,

— TF0 et TF1 correspondant aux timers 0 et 1,

sont positionnés sur S5P2 et traités sur le cycle machine suivant.

En revanche, l'indicateur TF2 du timer 2 est échantillonné comme actif durant la période S5P2 correspondant au cycle pendant lequel le compteur passe en débordement et la prise en compte et le traitement de la demande sont réalisés sur ce même cycle machine.

En conclusion de cette récapitulation, si une demande d'interruption est active et qu'aucune condition de blocage ne s'oppose au service de celle-ci, le processus de branchement au sous-programme de service prendra effet sur le cycle machine suivant l'échantillonnage ayant révélé la demande. L'exécution d'une instruction d'appel à un

sous-programme durant deux cycles machine, il faut donc compter un minimum de trois cycles machine complets entre l'instant où une demande d'interruption est détectée et l'instant où la première instruction du sous-programme de service va être exécutée.

A ce temps de réponse minimum il faut ajouter un délai variable si l'une des trois conditions de blocage s'oppose au service de l'interruption :

1 - Si une autre interruption de même niveau ou de niveau supérieur est en cours d'exécution, le temps de réponse à la nouvelle demande sera fonction de la durée de traitement du sous-programme en cours. Il n'est donc pas possible d'exprimer dans ce cas, un temps de réponse typique à la demande d'interruption.

2 - Si le cycle machine en cours n'est pas le dernier de l'instruction en cours d'exécution, le délai à ajouter au temps de réponse minimum n'excède pas trois cycles machine car les instructions les plus longues (MUL et DIV) ne nécessitent que quatre cycles. Le temps de réponse pour cette condition de blocage est donc au maximum de six cycles machine.

3 - Si l'instruction en cours d'exécution est un "**RETI**" ou une opération d'écriture dans l'un des registres IP ou IE, le délai à ajouter restera égal ou inférieur à cinq cycles machine : un cycle pour achever l'instruction en cours et de un à quatre cycles pour exécuter l'instruction supplémentaire prévue dans cette condition de blocage. En excluant le cas où les interruptions sont imbriquées, le temps de réponse à une demande d'interruption sera toujours supérieur à **trois** cycles machine et restera inférieur à **neuf** cycles machine.

Exemples d'utilisation

Exécution d'un programme en mode pas à pas

La structure du système d'interruption permet d'envisager une procédure d'exécution d'un programme en mode pas à pas.

Il faut se souvenir qu'une demande d'interruption ne peut pas être prise en compte durant le traitement d'une autre demande d'interruption de même niveau de priorité. La prise en compte et l'exécution de la nouvelle demande restent en suspend jusqu'à la fin du traitement en cours. La fin du sous-programme se termine par l'instruction RETI. Cette instruction provoque le retour au programme principal et autorise la prise en compte de la requête en attente. Après la réalisation du saut de retour, une instruction du programme principal est exécutée avant que l'exécution de la demande d'interruption en attente ait lieu.

En partant de ce principe, le constructeur INTEL propose une idée de programme qui permettrait d'exécuter un programme, instruction après instruction, et ce, au rythme d'impulsions électriques appliquées à une broche du microcontrôleur.

L'exemple utilise l'interruption extérieure INT0. Cette interruption correspond aux événements présents sur la broche P3.2. L'interruption INT0 doit être initialisée pour réagir sur état logique.

```
JNB  P3.2,$   ;attendre ici que la broche P3.2 soit à 1
JB   P3.2,$   ;attendre ici que P3.2 repasse à 0
RETI          ;et retourner exécuter une instruction
```

Cette suite de trois instructions doit être placée à partir de l'adresse 0003. La broche P3.2 est maintenue dans l'état logique 0 ce qui correspond à une demande permanente d'interruption. Le saut à l'adresse 0003 est donc assuré et l'indicateur IE0 correspondant à la source INT0 est automatiquement remis à 0. La première instruction assure une boucle d'attente sur la condition P3.2 = 1. Lorsque le niveau électrique présent sur P3.2 devient proche de 5 volts c'est la deuxième instruction qui assure une attente d'un nouvel état 0. Dès que la broche P3.2 repasse à 0, l'indicateur IE0 est positionné à 1 et donc une nouvelle demande d'interruption est enregistrée. Cette demande ne peut pas être servie avant l'exécution de l'instruction qui suivra l'instruction RETI. Une instruction du programme principal sera donc exécutée puis un saut à l'adresse 0003 assurera une synchronisation à l'impulsion électrique extérieure.

Cet exemple nécessite un circuit extérieur capable de générer des impulsions électriques (oscillateur ou bouton poussoir avec circuit antirebonds). Il est possible d'envisager une variante de ce programme.

La broche P3.2 est maintenue à 1 d'une façon définitive (état naturel d'un port d'entrée laissé en l'air), et l'on agit directement, par logiciel, sur l'indicateur IE0 :

```
LCALL  AFF_REG ; affiche état processeur par exemple
SETB   IE0     ; réamorce l'interruption
RETI           ; et retourne exécuter une instruction.
```

Gestion du port série sous interruption

L'amorce de programme présenté ci-dessous présente une gestion sous interruption du port série. Deux zones de la mémoire interne, de trente deux octets chacune, sont utilisées comme tampon (buffer) de réception et de transmission. Utiliser la communication revient alors à prélever, depuis le tampon de réception, les caractères reçus, ou à déposer les caractères à envoyer dans le tampon de transmission. Le sous-

programme d'interruption prend en charge la gestion des zones tampon. Le programme principal n'est pas très utile et sa simplicité peut rendre discutable la mise en œuvre d'une communication série gérée sous interruption. Il faut quand même remarquer que le programme assure l'envoi des chaînes de caractères **message, retour0 et retour1**. La gestion d'envoi de ces suites de caractères "occupe" le processeur. Durant ces périodes d'envoi, l'arrivée d'informations dans le registre de réception (SBUF) pourrait être perdue par phénomène d'écrasement (une première donnée étant remplacée par une autre avant même d'être prise en compte). La réception sous interruption évite ce risque. D'autre part la transmission sous interruption limite l'attente entre l'envoi de chaque donnée.

La gestion des zones tampon est partagée entre le sous-programme d'interruption et le programme principal. Chaque tampon est géré par deux pointeurs, l'un pour l'écriture, l'autre pour la lecture. La zone de mémoire interne, choisie pour recevoir la valeur de ces pointeurs, correspond à la deuxième banque de registres (banque 1). Les symboles définis correspondent en fait à un moyen d'adresser en mode direct ces registres sans que la banque soit obligatoirement active. Ainsi TXM_RD_PTR correspond à R1 et RCV_WR_PTR à R0 de la banque 1.

Etude du principe de gestion du tampon de réception :

Ce tampon occupe 32 octets à partir de l'adresse "**RCV_BUF**". Au départ les deux pointeurs RCV_WR_PTR et RCV_RD_PTR sont initialisés à la valeur de cette adresse, et le compteur RCV_COUNT prend la valeur 0. Lorsqu'un caractère est reçu, il est placé dans le tampon à l'adresse pointée par RCV_WR_PTR et le compteur RCV_COUNT est incrémenté de 1. Le pointeur RCV_WR_PTR est lui aussi incrémenté de 1 pour la mise en place du prochain caractère. La valeur du pointeur est testée afin de vérifier que la fin du tampon (TXM_BUF+B_LNG) n'est pas atteinte. Si cette extrémité est atteinte, le pointeur est replacé en début de tampon. Les nouvelles données vont donc recouvrir les anciennes, c'est-à-dire que le premier caractère reçu sera remplacé par le trente troisième. Mais entre temps, le programme principal a sans doute prélevé des caractères de ce tampon. Il les prélève grâce au pointeur RCV_RD_PTR depuis le sous-programme RS_IN. Ce sous-programme décrémente le compteur RCV_COUNT et déplace son pointeur de lecture RCV_RD_PTR. Bien que non prévu dans l'exemple, il est donc très facile de lever le doute sur le fait qu'un nouveau caractère en recouvre un précédent sans que ce dernier ai été récupéré par le programme principal : il suffit de tester la valeur du compteur RCV_COUNT et de s'assurer que ce compteur ne dépasse pas la valeur B_LNG (c'est-à-dire 32). Ce test peut être réalisé aussi bien dans le sous-programme RS_IN que dans le sous-programme d'interruption. Il faut noter que le sous-programme RS_IN peut subir d'autres modifications. Pour l'instant le programme principal appelle ce sous-programme sans savoir si un caractère est réellement arrivé dans le tampon de réception. Il y a donc blocage dans le sous-programme avec attente d'une réception.

Si l'application le nécessite, il est possible d'extraire le test de RCV_COUNT (deux premières lignes de RS_IN) et de faire ce test depuis le programme principal. Le résultat de ce test permet alors le choix entre récupérer un caractère ou faire autre chose...

Les directives d'assemblage DSEG, DS, BSEG, DBIT sont reconnues par l'assembleur INTEL ASM51. Elles sont décrites dans le chapitre 14.

```
B_LNG       EQU   32        ;LONGUEUR DES TAMPONS (BUFFER)
BANK_1      EQU   00001000B
```

LISTE DES VARIABLES PLACEES EN RAM INTERNE

```
            DSEG            AT 08H
;POINTEUR POUR ECRITURE DANS BUFFER DE RECEPTION = R0 BANQUE 1
RCV_WR_PTR: DS    1         ;à partir de l'adresse 08H
;POINTEUR POUR LECTURE DANS BUFFER DE TRANSMISSION
TXM_RD_PTR: DS    1         ;correspond à R1 BANQUE 1
;POINTEUR POUR LECTURE DANS BUFFER DE RECEPTION
RCV_RD_PTR: DS    1
;POINTEUR POUR ECRITURE DANS BUFFER DE TRANSMISSION
TXM_WR_PTR: DS    1
;COMPTEUR DE CARACTERES RECUS
RCV_COUNT:  DS    1
;COMPTEUR DE CARACTERES à ENVOYER
TXM_COUNT:  DS    1
```

ZONES DE MEMOIRE TAMPON
EN RAM INTERNE A PARTIR DE 30H

```
            DSEG  AT 30H
RCV_BUF:    DS    B_LNG       ;BUFFER DE RECEPTION
```

```
TXM_BUF :  DS     B_LNG      ;BUFFER DE TRANSMISSION
STACK :    DS     1          ;DEBUT DE LA PILE

;-------------------------------------------------------------------
;                    LISTE DES VARIABLES "BIT" *
;-------------------------------------------------------------------

           BSEG   AT (0)
FLAG_TXM : DBIT   1          ;=1 ENVOI AMORCE

;-------------------------------------------------------------------
;                    DEBUT DU CODE EXECUTABLE
;-------------------------------------------------------------------

           CSEG   AT 0        ; ADDRESSE RESET
           ORG    0H
FROID :
           LJMP   INI_SYSTEME ;APRES RESET SAUT A INI_SYSTEME

;-------------------------------------------------------------------
;          SOUS-PROGRAMME D'INTERRUPTION DU PORT SERIE
;                  PLACE A L'ADRESSE 23H
;          IL FAUT TESTER LES INDICATEURS TI ET RI POUR
;          IDENTIFIER LA CAUSE EXACTE DE L'INTERRUPTION
;-------------------------------------------------------------------

           ORG    23H         ; VECTEUR INTERRUPTION
SERIAL_INT:                   ; POUR PORT SERIE
           PUSH   PSW
           PUSH   ACC
           MOV    PSW,#BANK_1 ; activer banque de registres 1
           JBC    RI,RX_INT   ; SI RI=1 SAUT ET RI=0
TX_INT :
           MOV    A,TXM_COUNT ; GESTION DE L'ENVOI :
           JNZ    TX_INT0     ; un caractère est a envoyer ?
```

```
        CLR     FLAG_TXM        ; non alors désactiver indicateur
        SJMP    TX_IT2          ; et sortir
TX_INT0 :
        MOV     SBUF,@R1        ; OUI : envoyer car. vers SBUF
        INC     R1              ; mise à jour pointeur
        DEC     TXM_COUNT       ; et COMPTEUR
        CJNE    R1,#TXM_BUF+B_LNG,TX_IT2
        MOV     R1,#TXM_BUF     ; réinitialisation pointeur
TX_IT2: CLR     TI
        SJMP    IT_END
; L'interruption est due à un caractère reçu
RX_INT :
        MOV     @R0,SBUF        ; SI RI=1 placer caractère
        INC     R0              ; mise à jour pointeur
        INC     RCV_COUNT       ; et compteur de caractères
        CJNE    R0,#RCV_BUF+B_LNG,IT_END
        MOV     R0,#RCV_BUF     ; init pointeur
IT_END :
        POP     ACC
        POP     PSW             ; réactive BANQUE REG. 0
        RETI
```

ENTREE APRES RESET :
INITIALISATION DES VARIABLES, DE LA COMMUNICATION SERIE
ET AUTORISATION INTERRUPTION DEPUIS PORT SERIE

```
INI_SYSTEME :
        MOV     SP,#STACK
        MOV     RCV_WR_PTR,#RCV_BUF ; POINTEURS SUR DEBUT
        MOV     RCV_RD_PTR,#RCV_BUF ; DES BUFFERS
        MOV     TXM_WR_PTR,#TXM_BUF
        MOV     TXM_RD_PTR,#TXM_BUF
```

```
        MOV     TXM_COUNT,#0  ; COMPTEURS A 0
        MOV     RCV_COUNT,#0
        CLR     FLAG_TXM
```

INITIALISATION COMMUNICATION SERIE
QUARTZ DE 11.059 MHZ
LE TIMER 1 EST UTILISE COMME HORLOGE
LA VITESSE DE COMMUNICATION EST FIXEE A 9600 BAUDS

```
        MOV     SCON,#01011010B ;TI=1 POUR PREMIER ENVOI
        MOV     TMOD,#20H       ;8 BIT AUTO RELOAD TIMER 1
        MOV     TH1,#0FDH       ;VALEUR POUR 9600 BAUDS
        SETB    TR1             ;HORLOGE ACTIVE
```

MISE EN SERVICE DES INTERRUPTIONS POUR PORT SERIE

```
;VALIDE INT GENERALE ET PORT SERIE
        MOV     IE,#10010000B
```

PROGRAMME PRINCIPAL

```
MAIN :  MOV     DPTR,#MESSAGE   ; sortir un message
        CALL    OUT_STRG
MAIN_0 :
        CALL    RS_IN           ; attente d'un caractère
        CALL    RS_OUT          ; écho sur sortie série
        CJNE    A,#'1',PAS_UN   ; le caractère reçu correspond
        MOV     DPTR,#RETOUR1   ; au code ASCII de 1
        CALL    OUT_STRG        ; alors envoi d'un message
        SJMP    MAIN_0          ; et refaire
```

```
PAS_UN :  CJNE   A,#'0',PAS_0
          MOV    DPTR,#RETOUR0
          CALL   OUT_STRG
PAS_0 :   ........                ; suite du programme
          LJMP   MAIN_0           ; LOOP
```

SORTIE VERS PORT SERIE D'UNE CHAINE
POINTEE PAR DPTR ET TERMINEE PAR LE CODE 0

```
OUT_STRG :
          CLR    A
          MOVC   A,@A+DPTR
          CJNE   A,#0,MES1
          RET
MES1 :    CALL   RS_OUT
          INC    DPTR
          SJMP   OUT_STRG
```

ATTENTE D'UN CARACTERE DEPUIS BUFFER RECEPTION

```
RS_IN :   MOV    A,RCV_COUNT      ; test du compteur RCV_COUNT
          JZ     RS_IN            ; attendre ici si pas de caractère
RS_IN0 :
          DEC    RCV_COUNT
          MOV    R1,RCV_RD_PTR
          MOV    A,@R1
          INC    R1
          CJNE   R1,#RCV_BUF+B_LNG,RS_IN1
          MOV    R1,#RCV_BUF
RS_IN1 :
          MOV    RCV_RD_PTR,R1
          RET
```

DEPOT D'UN CARACTERE DANS BUFFER DE TRANSMISSION

```
RS_OUT:
            MOV     B,A                     ; sauvegarde caractère à envoyer
RS_OUT3:
            MOV     A,TXM_COUNT
            CJNE    A,#B_LNG-1,RS_OUT2      ; BUFFER plein ?
RS_OUT2:
            JNC     RS_OUT3                 ; ALORS attendre
            MOV                             ; R1, TXM_WR_PTR ; SINON
            MOV     @R1,B                   ; mise en place du caractère
            INC     R1                      ; dans buffer et mise à jour
            INC     TXM_COUNT               ; nombre caractères en attente
            CJNE    R1, # TXM_BUF+B_LNG,RS_OUT0
            MOV     R1, # TXM_BUF           ; mise à jour pointeur
RS_OUT0:
            MOV     TXM_WR_PTR,R1
            MOV     A,B
            JB      FLAG_TXM,RS_OUT1        ; saut si envoi déjà amorcé
            SETB    FLAG_TXM
            CLR     EA                      ; provoquer interruption
            SETB    TI                      ; pour envoi premier
            SETB    EA                      ; caractère
RS_OUT1:
            RET
;LISTE DES MESSAGES
MESSAGE:        DB ' SYSTEME PRET!!! ',0
RETOUR1:        DB ' VALEUR 1 ',13,10,0
RETOUR0:        DB ' VALEUR 0 ',13,10,0
                END
```

13

Processeur Booléen

Définition du terme "processeur booléen"

Le terme de processeur booléen a été évoqué dans les chapitres 2 et 6. Rappelons simplement que cette expression désigne la possibilité d'utiliser l'unité centrale 8 bits comme une unité de traitement au niveau du bit. Il est donc possible de tester une entrée, de positionner une sortie sans modifier le reste de l'ensemble 8 bits formé par un port. Cet adressage au niveau du bit caractérise aussi une partie de la mémoire interne. Ce type d'adressage est exploité par une partie du jeu d'instructions (tableaux 13.1a et 13.1b). Le processeur booléen utilise comme "registre accumulateur" l'indicateur de report C.

Mnémonique	Opération	Cycle
ANL C, bit	C=C and bit	2
ANL C,/bit	C=C and not bit	2
ORL C, bit	C=C or bit	2
ORL C,/bit	C=C or not bit	2
MOV C,bit	C=bit	1
MOV bit,C	bit=C	2
CLR C	C=0	1
CLR bit	bit=0	1
SETB C	C=1	1
SETB bit	bit=1	1
CPL C	C=not C	1
CPL bit	bit=not bit	1

Tableau 13.1a : Instructions boléennes

Mnémonique	Opération	Cycle
JC REL	saut si C=1	2
JNC rel	saut si C=0	2
JB bit, rel	saut si bit=1	2
JNB bit,rel	saut si bit=0	2
JBC bit,rel	saut si bit=1 & bit=0	2

Tableau 13.1 b : Sauts conditionnels utilisant l'adressage au niveau du bit

Intérêt de l'utilisation du processeur booléen

Afin d'apprécier l'utilité du processeur booléen, on se propose de traiter un exemple très simple résumé par l'équation logique :

$$S = \bar{e}$$

où S représente la broche P1.0, et e la broche P1.1

Dans une approche classique (sans utiliser le processeur booléen), ce problème simple doit se décomposer de la façon suivante :

1. Lire l'ensemble 8 bits du port P1 ;
2. isoler l'état de la broche P1.1 ;
3. tester l'état du bit isolé pour définir l'état de S ;
4. replacer sur le port cet état sans modifier le reste du port

Ce qui se traduit par la suite d'instructions

```
        MOV  A,P1           ; ETAT DU PORT DANS ACCUMULATEUR
        ANL  A,#00000010B   ; MASQUE POUR P1.1
        JZ   E_ZERO         ; SI A=0 E=0
        ANL  P1,#11111110B  ; SINON E=1 D'OU S=0
        SJMP SORTIE
E_ZERO: ORL  P1,#00000001B  ; E=0 DONC S=1
SORTIE: .....
```

L'utilisation du processeur booléen permet de réduire considérablement le traitement de cet exemple :

```
MOV     C,P1.1              ; LIRE L'ENTREE
CPL     C                   ; COMPLEMENTER L'ETAT
MOV     P1.0,C              ; ACTUALISER LA SORTIE
```

On remarquera que la première solution utilise la possibilité de réaliser des opérations logiques directement sur un registre de port. Cette façon de faire est rarement possible sur un microprocesseur classique. Un transit par l'accumulateur est alors nécessaire ce qui alourdit encore plus le traitement du problème.

Programmation d'un circuit de logique combinatoire

L'équation logique

$$Q = \left(u\,(v + w)\right) + (x.\overline{y}) + \overline{z}.$$

peut se traduire par le logigramme de la figure 13.2. D'un point de vue technologique, il est possible de réaliser une telle fonction à partir de circuits intégrés (TTL ou CMOS) ou bien à partir d'un ensemble de contacts et de relais (figure 13.3.). Bien que facultatifs, les relais auxiliaires KA1 et KA2 représentent des résultats intermédiaires du calcul. Leur utilisation simplifient la compréhension du fonctionnement.

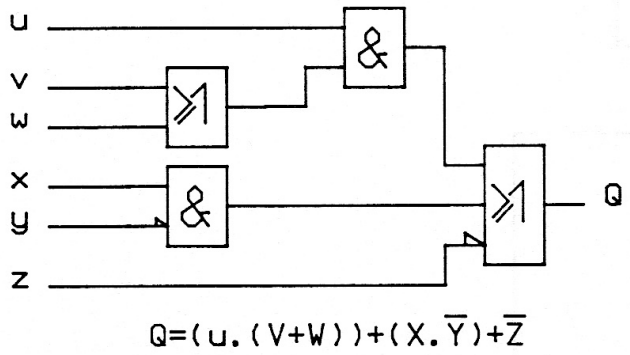


Figure 13.2 : Logigramme du circuit à programmer

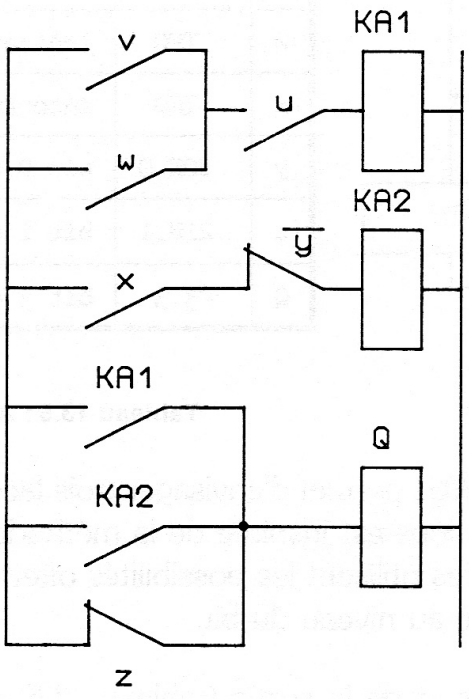


Figure 13.3 : Schéma à relais

Il est aussi possible d'envisager de remplir cette même fonction par programmation à partir d'un microcontrôleur. Une approche classique de programmation conduit à établir un organigramme tel que celui de la figure 13.4.

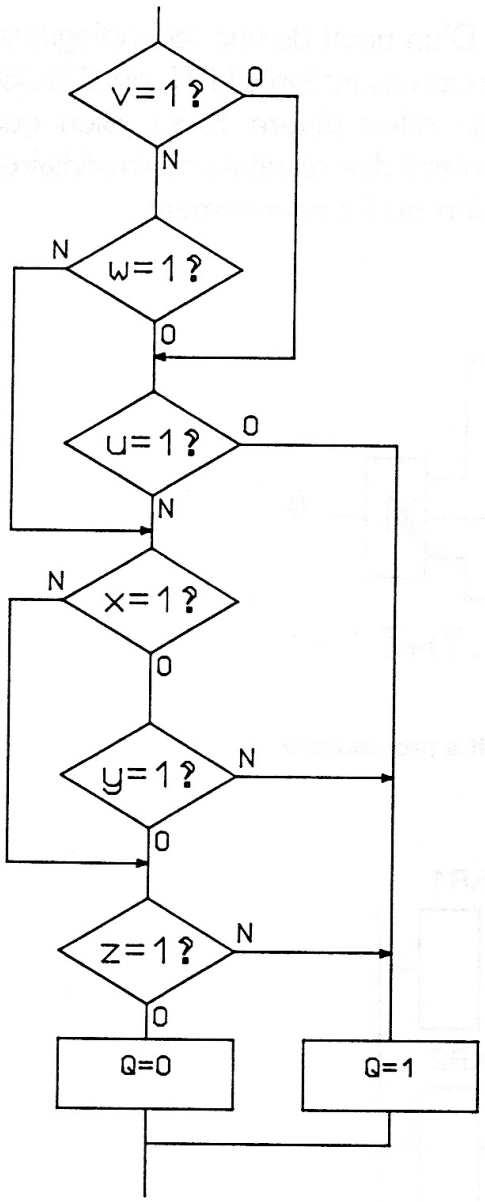


Figure 13.4 : Organigramme

u	P1.1	bit 1 du port P1
v	p2.2	bit 2 du port P2
w	TF0	indicateur de débordement TIMER 0
x	IE0	interruption externe 0
y	20H.0	bit 0 de l'octet d'adresse 20H
z	21H.1	bit 1 de l'octet 21H
Q	P3.3	bit 3 du port P3

Tableau 13.5 : Affectation entrées et sorties

Le jeu d'instructions du 8051 permet d'envisager trois façons de résoudre ce problème. La première de ces solutions est inspirée de la méthode utilisée sur des processeurs classiques, les deux autres utilisent les possibilités offertes par le processeur booléen et son mode d'adressage au niveau du bit.

L'affectation des entrées et de la sortie (tableau 13.5) reste la même dans les trois exemples. Elle est choisie arbitrairement.

Solution utilisant un adressage au niveau de l'octet

Le test de l'état d'une variable d'entrée est réalisé par opération de masquage et un saut est exécuté selon le résultat de cette opération (nul ou non nul). Les différents masques sont établis en fonction de la position de la variable d'entrée dans l'octet auquel elle appartient. Le programme respecte l'organigramme de la figure 13.4.

```
SORTIE   EQU   22H          ; représente l'état des 8 bits de P3
TST_V:   MOV   A,P2         ; lecture de V
         ANL   A,#00000100B ; masque pour isoler P2.2
         JNZ   TST_U        ; V=1 alors tester u
         MOV   A,TCON       ; lire w (TF0)
         ANL   A,#00100000B ; isoler TF0 dans TCON
         JZ    TST_X        ; si w= 0 alors tester X
TST_U:   MOV   A,P1         ; lecture de u dans P1
         ANL   A,#00000010B ; isoler u=P1.1
         JNZ   SET_Q        ; si u=1 Q=1 et fin
TST_X:   MOV   A,TCON       ; lire x dans TCON
         ANL   A,#00001000B ; isoler x=IE0
         JZ    TST_Z        ; si x=0 alors tester z
         MOV   A,20H        ; sinon lire y dans 20H
         ANL   A,#00000001B ;       isoler y
         JZ    SET_Q        ; si y=0 faire Q=1
TST_Z:   MOV   A,21H        ; sinon tester z
         ANL   A,#00000010B
         JZ    SET_Q        ; si z=0 faire Q=1
CLR_Q:   MOV   A,SORTIE     ; mettre Q à 0 à partir
         ANL   A,#11110111B ; de SORTIE
         SJMP  OUT_Q
SET_Q:   MOV   A,SORTIE     ; mise à 1 de Q
         ORL   A,00001000B
OUT_Q:   MOV   SORTIE,A     ; mise à jour de SORTIE
         MOV   P3,A         ; et du port P3
```

Solution utilisant les instructions de test de bit

Ce deuxième programme utilise les instructions du tableau 13.1b. Sa conception découle directement de l'organigramme. Il offre donc une bonne lisibilité. De plus, l'absence de l'emploi de masque servant à isoler les variables permet de limiter les risques d'erreurs d'écriture. On passe de 25 lignes de programme à 9 lignes pour le même résultat !

```
TST_V :   JB     V,TST_U      ; si V=1 tester U
          JNB    W,TST_X      ; sinon si W=0 tester X
TST_U :   JB     U,SET_Q      ; si U=1 faire Q=1.
TST_X :   JNB    X,TST_Z      ; si X=0 tester Z
          JNB    Y,SET_Q      ; sinon si Y=0 faire Q=1.
TST_Z :   JNB    Z,SET_Q      ; si Z=0 alors faire Q=1
CLR_Q :   CLR    Q            ; sinon Q=0
          SJMP   SUITE        ; et fin
SET_Q :   SETB   Q            ;
SUITE : .....
```

Le temps d'exécution est variable : il dépend de l'état des variables d'entrées. Il est au minimum de 7 cycles machine et au maximum de 13 cycles machine.

Utilisation des instructions booléennes

Cette troisième version utilise les instructions du tableau 13.1a. Le temps d'exécution est constant (16 cycles machine) car aucun saut conditionnel n'est utilisé. Si les deux premières versions prenaient l'organigramme comme modèle, cette version est calquée sur l'écriture de l'équation logique.

On remarquera l'utilisation du bit F0 du registre PSW utilisé comme mémoire d'un résultat partiel. Cette utilisation peut être comparée à l'utilisation des relais auxiliaires KA1 et KA2 dans le schéma de la figure 13.3.

```
MOV    C,v          ;
ORL    C,w          ; OU logique entre v et w
ANL    C,u          ; C = u(v+w)
MOV    F0,C         ; ce résultat est placé dans F0.
```

```
MOV     C,x         ;
ANL     C,/y        ; ET logique entre x et complément de y
ORL     C,F0        ;
ORL     C,/z        ; traitement du OU à 3 entrées
MOV     Q,C         ; résultat vers sortie Q
```

Dans le cas où la rapidité de traitement n'est pas une exigence fondamentale, la conception d'un tel programme reste aisée même pour des équations logiques complexes : il suffit de généraliser l'emploi de bits mémoires afin de stocker des résultats intermédiaires. Si le temps semble être compté, il faut quand même avoir présent à l'esprit qu'une réalisation basée sur une technologie à relais électromagnétique est loin d'être instantanée ! Le temps de collage d'un très bon relais reste supérieur à la milliseconde et un capteur d'information, pour peu qu'il soit doté de part son principe, d'un contact mécanique, possédera un temps de réponse non négligeable.

L'exemple qui suit ne cherche donc pas l'optimisation même si celle-ci est parfois évidente. Le programme est écrit d'une façon systématique comme pourrait le faire un compilateur chargé de traduire une équation logique en code source.

Soit à réaliser l'équation logique

$$S = a\left(b + \overline{c}\,(v + x)\right) + e\,(y + \overline{z})$$

L'ordre de résolution doit prendre en compte le degré de mise en facteur. On commence par le degré de parenthèse le plus profond.

où t1 représente une mémoire temporaire

$$S = a\left(b + \overline{c}\,(t1)\right) + e\,(y + \overline{z})$$

puis faire $t1 = \overline{c}\,.\,t1$

pour arriver à $S = a\,(b + t1) + e\,(y + \overline{z})$

etc. pour arriver finalement à $S = t1 + t2$.

t1 et *t2* sont deux bits situés dans l'octet de RAM interne d'adresse 20H (début de la zone adressable au niveau du bit).

```
t1      bit 0           ;bit 0 de l'octet 20H
t2      bit 1           ;bit 1 de l'octet 20H
        MOV     C,x
```

```
ORL     C,v
MOV     t1,C        ; t1 = x+v
MOV     C,t1
ANL     C,/c
MOV     t1,C        ; t1 = /c(x+v)
MOV     C,t1
ORL     C,b
MOV     t1,C        ; t1 = a+/c(x+v)
MOV     C,y         ; OU logique entre y
ORL     C,/z        ; et le complément de z
MOV     t2,C        ; placé dans t2
MOV     C,t1
ANL     C,a
MOV     t1,C        ; t1 = a . t1
MOV     C,t2
ANL     C,e
MOV     t2,C        ; t2 = e . t2
MOV     C,t1
ORL     C,t2
MOV     S,C         ; définir l'état de sortie
```

Cette suite d'instructions peut bien sûr être simplifiée. Il est inutile de laisser les suites

```
MOV     t1,C
MOV     C,t1
```

Programmation d'un circuit de logique séquentielle

La représentation du type GRAFCET de niveau 2 va être utilisée pour décrire les exemples à traiter. Le GRAphe Fonctionnel de Commande Etape Transition constitue un modèle graphique basé sur les notions d'étape et de réceptivité associées aux transitions. Une étape constitue une situation invariable de la commande de l'automatisme. Les actions associées à une étape ne sont assurées que si l'étape est dite "active". Le grafcet est constitué d'une alternance obligatoire d'étapes et de transitions.

Une transition ne peut être valide que si toutes les étapes, immédiatement précédentes qui lui sont reliées, sont actives.

Une transition est franchissable si la transition est valide et que la réceptivité qui lui est associée est vraie.

Une réceptivité s'exprime sous forme d'une équation logique précisant les relations de prise en compte des entrées de la commande.

Le franchissement d'une transition provoque simultanément :

— la désactivation de toutes les étapes immédiatement précédentes reliées à cette transition ;

— l'activation de toutes les étapes immédiatement suivantes reliées à cette transition.

Dernier rappel : Pour qu'un grafcet puisse évoluer normalement, il est nécessaire qu'au moins une étape soit active à l'initialisation. C'est en général le rôle de l'étape initiale représentée par une étape dont les côtés sont doublés (voir l'étape 0 du premier exemple).

Programmation d'une séquence simple

L'exemple de la figure 13.6 présente une séquence simple de trois étapes (0, 1 et 2). Ces trois étapes sont séparées par trois transitions auxquelles sont associées les réceptivités a(b+c), x et y. L'étape 1 provoque l'activation des sorties EV+ et M, l'étape 2 active les sorties EV— et R. Il est sous-entendu que toute sortie non activée est forcée à 0.

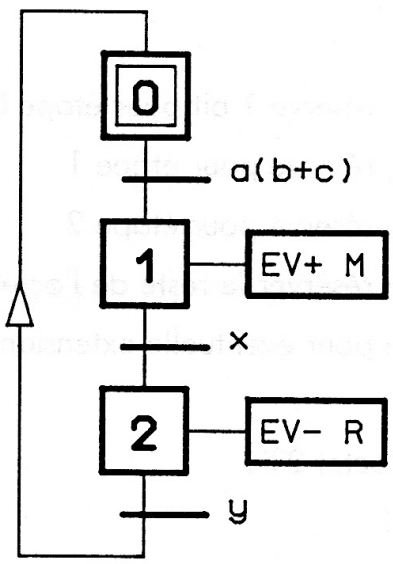


Figure 13.6 : séquence de 3 étapes.

La conception générale du programme sera organisée selon quatre phases :

1. lire les entrées ;
2. définir l'état des réceptivités ;
3. actualiser l'état des étapes ;
4. actualiser l'état des sorties.

Au niveau programmation, des bits de la RAM interne vont être associés aux étapes, et aux réceptivités. Ces bits sont situés dans la zone de mémoire interne adressable au niveau du bit.

Le programme qui suit est donné comme principe de base. Il ne tient pas compte de certaines sécurités à prévoir :

— réinitialisation après coupure d'alimentation ;

— fiabilité de l'actualisation des sorties ;

— filtre des états parasites pouvant être lus sur les entrées ;

— surveillance du bon déroulement du programme ;

etc .

BIT MAP départ octet adresse 20H

```
                BSEG  AT (0)
;
ETAPE0 :        DBIT 1      ; réserve 1 bit pour étape 0
ETAPE1 :        DBIT 1      ; réserve pour étape 1
ETAPE2 :        DBIT 1      ; réserve pour étape 2
                DBIT 1      ; réserver le reste de l'octet
                DBIT 1      ; pour éventuelle extension
;
; laisser libre la fin de l'octet 20H
; et passer à l'octet 21H
                BSEG  AT (8)
;
```

```
RECEP1 :    DBIT 1                  ; résultat réceptivité a(b+c)
RECEP2 :    DBIT 1                  ; réceptivité x
RECEP3 :    DBIT 1                  ; réceptivité y
            DBIT 1                  ; pour extension
            DBIT 1
            BSEG    AT (0FH)
; OCTET 22H = IMAGE du port P1
; toutes les variables d'entrée sont câblées sur P1
; tel que a=P1.0 b=P1.1 c=P1.2 x=P1.3 et y=P1.4
;
IN_a :      DBIT 1
IN_b :      DBIT 1
IN_c :      DBIT 1
IN_x :      DBIT 1
IN_y :      DBIT 1
            DBIT 1
;Segment code à partir de l'adresse 0 dans EPROM
            CSEG    AT (0)
;
INIT :      MOV     20H,#01     ; seule l'étape 0 est active
            MOV     21H,#0      ; toutes les réceptivités à 0
;lecture des entrées
RD_IN :     MOV     22H,P1      ; copie du port P1 dans RAM.
;calcul des
réceptivités
;
            MOV     C,IN_c      ; faire c+b
            ORL     C,IN_b      ; C=c+b
            ANL     C,IN_a      ; C=a(b+c)
            MOV     RECEP1,C    ; ranger réceptivité a(b+c)
;
            MOV     C,IN_x      ; idem pour réceptivité 2
            MOV     RECEP2,C
```

```
;
            MOV     C,IN_y      ; idem pour réceptivité 3
            MOV     RECEP3,C
;
;Actualiser les bits ETAPE
            MOV     C,ETAPE0    ; si ETAPE0 et RECEP1 =1
            ANL     C,RECEP1    ; alors ETAPE0=0 et ETAPE1=1
            MOV     ETAPE1,C    ; sinon laisser dans l'état
            CPL     C
            MOV     ETAPE0,C
; étape 1
            MOV     C,ETAPE1    ; si ETAPE1 et RECEP2 =1
            ANL     C,RECEP2    ; alors ETAPE1=0 et ETAPE2=1
            MOV     ETAPE2,C    ; sinon laisser dans l'état
            CPL     C
            MOV     ETAPE1,C    ;
; étape 2
            MOV     C,ETAPE2    ; si ETAPE2 et RECEP3 =1
            ANL     C,RECEP3    ; alors ETAPE2=0 et ETAPE0=1
            MOV     ETAPE0,C    ; sinon laisser dans l'état
            CPL     C
            MOV     ETAPE2,C    ; le "rebouclage" est réalisé
;Actualiser les sorties à partir des bits "ETAPE"
            JNB     ETAPE0,OUT1
            CLR     P3.0        ; P3.0 = EV— = 0
            CLR     P3.1        ; P3.1 = R = 0
            CLR     P3.2        ; P3.2 = EV+ à 0 par sécurité
            CLR     P3.3        ; P3.3 = M à 0 par sécurité
OUT1 :      JNB     ETAPE1,OUT2
            SETB    P3.2        ; P3.2 = EV+
            SETB    P3.3        ; P3.3 = M
```

```
OUT2 :    JNB     ETAPE2,FIN
          CLR     P3.2        ; si ETAPE2
          CLR     P3.3        ; RAZ sortie EV+ et M
          SETB    P3.0        ; puis EV— et R = 1
          SETB    P3.1
; Puis recommencer les quatre phases
FIN :     JMP     RD_IN
;
```

Divergence et convergence en ET

Le grafcet de la figure 13.7, bien qu'incomplet (aucune sortie n'est affectée aux 7 étapes), présente l'utilisation d'une divergence et d'une convergence en ET. La divergence en ET se traduit par la règle suivante :

Si l'étape 0 est active et si la réceptivité a est vraie alors l'étape 0 est désactivée et les deux étapes 1 et 4 sont activées.

Puis les deux branches évoluent indépendamment l'une de l'autre. Les étapes 3 et 5 constituent des étapes d'attente. Pour que l'étape 6 soit activée, il faut que les étapes 3 et 5 soient actives et que la réceptivité x soit vraie.

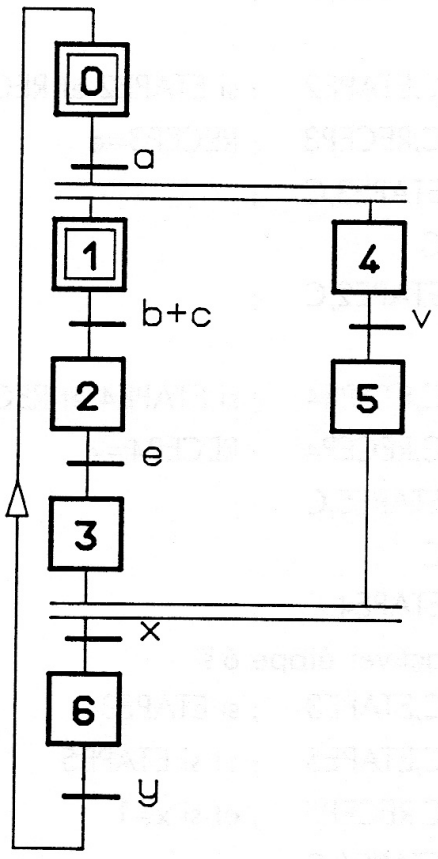


Figure 13.7

L'actualisation des bits "ETAPE" de ce grafcet est donnée ci-dessous. On remarquera que l'ordre de traitement n'a pas une importance primordiale. On peut choisir par exemple un traitement ordonné par branche en adoptant une "priorité" de la gauche vers la droite.

```
.....
;Actualiser les bits ETAPE
;traitement de la divergence en ET
            MOV   C,ETAPE0   ; si ETAPE0 et RECEP1 =1
            ANL   C,RECEP1   ; RECEP1=a
            MOV   ETAPE1,C   ; et ETAPE1=1
            MOV   ETAPE4,C   ; et ETAPE4=1
            CPL   C
            MOV   ETAPE0,C   ; et ETAPE0=0 ; activer étape 2 ?
            MOV   C,ETAPE1   ; si ETAPE1 et RECEP2 =1
            ANL   C,RECEP2   ; RECEP2=b+c
            MOV   ETAPE2,C   ;
            CPL   C
            MOV   ETAPE1,C   ;
; activer étape 3 ?
            MOV   C,ETAPE2   ; si ETAPE2 et RECEP3 =1
            ANL   C,RECEP3   ; RECEP3=e
            MOV   ETAPE3,C   ;
            CPL   C
            MOV   ETAPE2,C   ;
; activer étape 5 ?
            MOV   C,ETAPE4   ; si ETAPE4 et RECEP4 =1
            ANL   C,RECEP4   ; RECEP4=v
            MOV   ETAPE5,C   ;
            CPL   C
            MOV   ETAPE4,C   ;
; convergence en ET activer étape 6 ?
            MOV   C,ETAPE3   ; si ETAPE3
            ANL   C,ETAPE5   ; et si ETAPE5
            ANL   C,RECEP5   ; et si x=1
            MOV   ETAPE6,C   ;
            CPL   C
```

```
            MOV   ETAPE3,C   ; désactiver ETAPE3
            MOV   ETAPE5,C   ; et ETAPE5
; activer étape 0 ?
            MOV   C,ETAPE6   ; si ETAPE6
            ANL   C,RECEP6   ; et si y=1
            MOV   ETAPE0,C   ; ETAPE0=1
            CPL   C          ;
            MOV   ETAPE6,C   ; ETAPE6=0
.....
```

Divergence et convergence en OU

Une divergence en OU traduit la notion de choix d'une séquence d'opérations parmi d'autres séquences. La figure 13.8 présente une divergence en OU entre la séquence ETAPE 1, ETAPE 2 et la séquence ETAPE 3, ETAPE 4. Si une séquence est exécutée, l'autre ne le sera pas.

Si l'étape 0 est active et que la réceptivité a est vraie, alors : l'étape 0 est désactivée et l'étape 1 est activée.

Si l'étape 0 est active et que la réceptivité b est vraie, alors :

l'étape 0 est désactivée et l'étape 3 est activée.

On observe que la désactivation de l'étape 0 implique qu'une seule des deux conditions pourra être remplie.

La vérification de ces 2 conditions ne pouvant être simultanée, l'ordre du traitement prend ici une grande importance. Il est nécessaire de fixer une priorité dans le cas où les réceptivités a et b seraient vraies en même temps. Dans le programme qui suit, la priorité donnée est celle d'une lecture du grafcet allant de gauche vers la droite.

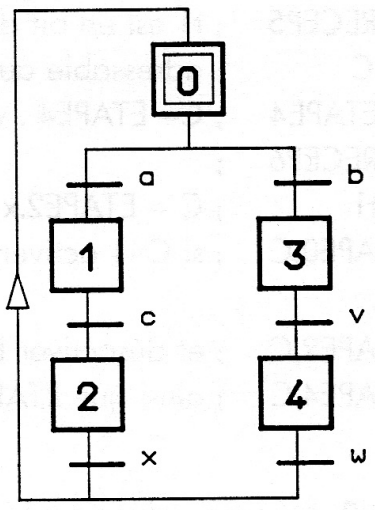


Figure 13.8

```
;traitement de la divergence en OU
        MOV   C,ETAPE0    ; si ETAPE0 et RECEP1 =1
        ANL   C,RECEP1    ; RECEP1=a
        MOV   ETAPE1,C    ; et ETAPE1=1
        CPL   C
        MOV   ETAPE0,C    ; et ETAPE0=0
; activer étape 2 ?
        MOV   C,ETAPE1    ; si ETAPE1 et RECEP3 =1
        ANL   C,RECEP3    ; RECEP3=c
        MOV   ETAPE2,C    ;
        CPL   C
        MOV   ETAPE1,C    ;
; activer étape 3 dans le cas où a= 0
        MOV   C,ETAPE0    ; si ETAPE0 et RECEP2 =1
        ANL   C,RECEP2    ; RECEP2=b
        MOV   ETAPE3,C    ;
        CPL   C
        MOV   ETAPE0,C    ; et ETAPE0=0
; activer étape 4 ?
        MOV   C,ETAPE3    ; si ETAPE4 et RECEP4 =1
        ANL   C,RECEP4    ; RECEP4=v
        MOV   ETAPE3,C    ;
        CPL   C
        MOV   ETAPE3,C    ;
; traitement de la convergence en OU
        MOV   C,ETAPE2    ; t1= ETAPE2 . x
        ANL   C,RECEP5    ; t1 est un bit de la zone
        MOV   t1,C        ; adressable au niveau du bit
        MOV   C,ETAPE4    ; C= ETAPE4 . w
        ANL   C,RECEP6    ;
        ORL   C,t1        ; C = ETAPE2.x + ETAPE4.w
        MOV   ETAPE0,C    ; si C=1 activer ETAPE0
        CPL   C
        MOV   ETAPE2,C    ; et désactiver ETAPE2
        MOV   ETAPE4,C    ; ainsi que ETAPE4
```

On désactive les étapes 2 et 4 bien qu'une seule ait été active. Le traitement proposé pour ce type de problème peut bien sûr être conçu de façon différente.

14

Aide au développement matériel et logiciel

Élaboration d'un projet

On posera ici comme à priori que le projet à réaliser, le sera à partir de la mise en œuvre d'un microcontrôleur en tant que commande du système.

Les différentes analyses réalisées à partir du cahier des charges, et formant les bases d'une bonne conception, ne seront pas développées dans ce chapitre.

Notons simplement que :

1 — Le cahier des charges du projet doit exprimer le plus clairement possible :

— les fonctions et les services attendus ;

— les performances souhaitées ;

— les limites tolérées ;

— les contraintes de l'environnement.

2 — Les outils d'analyses choisis doivent permettre de :

— définir la structure matérielle ;

— préparer l'organisation du logiciel ;

3 — Toute phase de conception doit s'accompagner d'une documentation bien souvent utile au moment des phase de test et de recherche d'erreurs

Choix de l'organisation matérielle

Définir la structure matérielle revient à :

— décrire l'interfaçage nécessaire entre la partie **commande** (le microcontrôleur) et la partie **opérative** du système ;

— préciser les relations avec l'environnement au système.

L'interfaçage avec la partie OPERATIVE peut s'exprimer en terme de :

— nombre d'entrées ;

— nature des signaux à traiter ;

— nombre de sorties.

Les relations avec l'environnement peut nécessiter :

— des entrées (clavier de commande, capteurs etc...)

— des sorties (afficheurs, voyants, etc...)

— un support normalisé d'échange d'informations (port série ou parallèle etc...)

Une fois l'analyse de conception amorcée tant au plan logiciel qu'au plan des besoins matériels il est possible d'effectuer des choix qui conduiront à la réalisation du support matériel de l'application.

Programme interne ou externe ?

Les éléments à prendre en compte pour donner une réponse à cette question peuvent se résumer de la façon suivante :

Si le projet doit être réalisé industriellement en production de masse la solution est alors toute trouvée ! Faites programmer vos microcontrôleurs lors de leur fabrication par le constructeur.

Pour une production unitaire ou de petite série, il reste l'alternative d'utiliser :

— soit, un microcontrôleur avec EPROM intégrée ;

— soit, un microcontrôleur sans mémoire programme, complété par une EPROM standard câblée en externe.

Microcontrôleur avec EPROM intégrée

Principaux avantages :

— gain d'encombrement sur le circuit imprimé ;

— tous les ports d'entée/sorties restent disponibles à condition que l'application ne nécessite pas de mémoire externe de données ;

— possibilité de protéger le programme contre les copies illicites.

Inconvénients :

— prix de revient souvent plus élevé que la combinaison microcontrôleur + EPROM ;

— impossibilité d'utiliser un simulateur d'EPROM lors des phases de tests de l'application ;

— nécessité de posséder un "programmateur d'EPROM" supportant l'adaptation au microcontrôleur utilisé.

L'utilisation d'un outil au coût plus élevé, comme l'émulateur, est donc nécessaire pour réaliser les mises au point de l'application.

Microcontrôleur + EPROM externe

Les avantages et inconvénients de cette solution découlent de l'analyse du cas précédent :

Avantages :

— prix de revient ;

— possibilité de moduler la taille de la mémoire programme ;

— un simulateur d'EPROM peut suffire lors des phases de tests de l'application ;

— utilisation d'un "programmateur d'EPROM" classique pour la mise en place du code ;

— mise à jour plus économique car l'échange standard se limite au remplacement de l'EPROM.

Inconvénients :

— protection du programme impossible ;

— mobilisation de deux ports en guise de bus externes ;

— augmentation relative de la complexité du câblage.

Les deux derniers inconvénients ne sont plus à prendre en compte si une mémoire externe de données s'avère nécessaire dans le cadre de l'application. Avant d'envisager une telle mémoire de données, il convient d'utiliser au mieux la mémoire interne.

Organisation de la RAM interne

Définir l'utilisation de la RAM interne revient à réserver des zones mémoires à des fonctions propres à l'application envisagée. L'affectation détaillée des 128 octets (ou 256 pour les versions 8052) reste donc une suite de cas particuliers. Il est possible malgré tout de suivre quelques règles d'utilisation.

Zone mémoire 0 à 1FH :

Cette zone correspond aux quatre banques de registres. La banque 0, active à l'initialisation, peut conserver son rôle dans le traitement principal du programme. Les trois autres banques peuvent être consacrées aux sous-programmes d'interruption. Il est alors possible de préserver l'état des 8 registres en cours d'utilisation, et d'activer 8 autres registres dédiés au traitement de l'interruption. Une banque inutilisée peut être considérée comme huit octets de mémoire dont l'utilisation sera à définir.

Zone mémoire 20H à 2FH :

Les seize octets de cette zone étant adressables au niveau du bit, elle devra être utilisée en priorité comme zone de variables binaires (indicateurs d'état de traitement, image d'une entrée ou d'une sortie logique etc...)

Zone mémoire 30H à 7FH :

Cette troisième partie de la mémoire peut être utilisée pour placer les variables de format 8 bits (ou plus) du programme. Elle peut aussi être utilisée comme tampon de réception et d'émission de la communication série.

Il faut se souvenir que le pointeur de pile est initialisé de la valeur 07. La pile d'origine se situe donc sur la banque 1 de registres. Il est souvent conseillé de modifier la valeur du pointeur SP et de placer la pile au dessus des autres variables de l'application. Si l'on utilise une version du microcontrôleur dotée de 256 octets, la pile peut empiéter sur la zone mémoire située au delà de 7FH. Rappelons d'ailleurs que cette zone n'est adressable qu'en mode indirect, ce qui convient parfaitement à la gestion de la pile. La figure 14.1 résume une utilisation classique de la RAM interne.

C'est après avoir attribué une fonction à chaque zone de la RAM interne, qu'il sera possible d'estimer la nécessité de compléter cette mémoire interne de données par un composant mémoire externe. L'éventualité d'étendre la mémoire de données en mode

externe s'accompagne de la réquisition du port P0, d'une partie (voir de l'ensemble) du port P2 et de l'adjonction d'un circuit de démultiplexage. Cette remarque est sans objet si la solution microcontrôleur + EPROM externe est retenue.

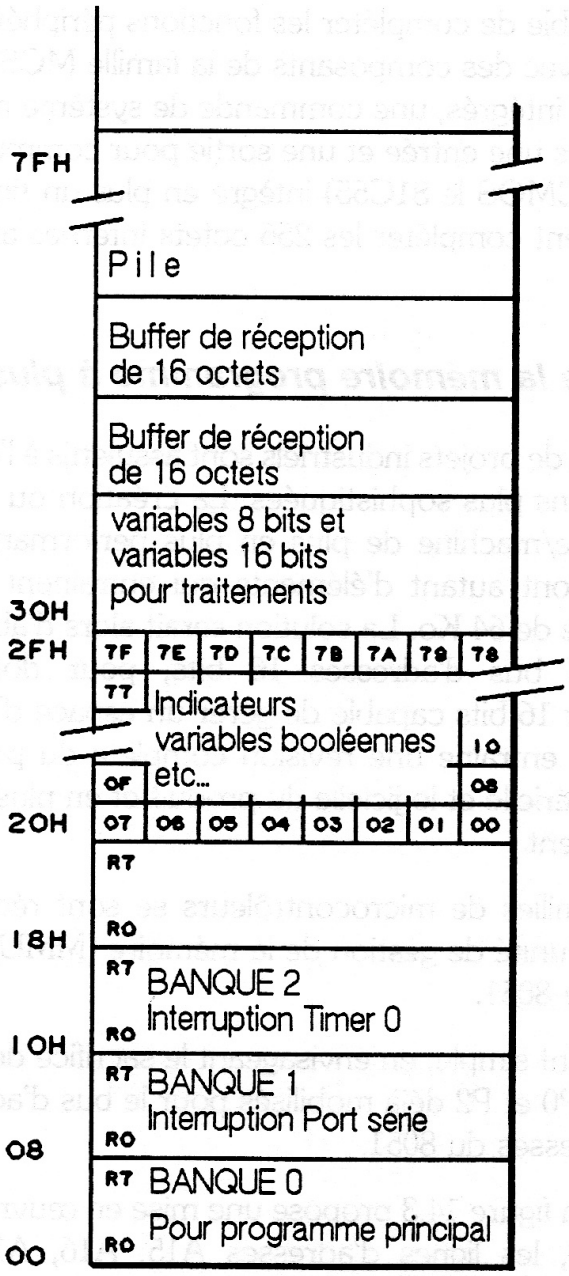


Figure 14.1 : Utilisation de la RAM interne

Composants de la famille MCS-80/85

La commercialisation des microprocesseurs 8 bits 8080 et 8085 par la société INTEL fut accompagnée par la mise sur le marché d'un bon nombre de composants périphériques qui répondaient au souci de créer autour d'une unité centrale un environnement minimum et ce avec un minimum de composants. La famille 8051 a

hérité d'un bon nombre de concepts déjà utilisés sur le microprocesseur 8085 :
— notion de bus multiplexe entre adresse et donnée
— commande de la périphérie par des signaux de même nature (ALE, $\overline{RD}$, $\overline{WR}$ etc.).

Il est donc possible de compléter les fonctions périphériques d'un microcontrôleur de la famille 8051 avec des composants de la famille MCS 80/85. La figure 14.2 propose, en deux circuits intégrés, une commande de système ne comportant pas moins de 41 lignes d'E/S (plus une entrée et une sortie pour communication série). Le circuit 8155 (ou sa version CMOS le 81C55) intègre en plus un timer de 14 bits et 256 octets de RAM qui viennent compléter les 256 octets internes au 87C52.

Extension de la mémoire programme à plus de 64 Ko

Un bon nombre de projets industriels sont assujettis à l'évolution de leur version initiale pour des versions plus sophistiquées. La création ou l'amélioration de fonctions, un dialogue homme/machine de plus en plus performant, la commande de nouveaux périphériques sont autant d'éléments qui entraînent rapidement un programme à dépasser la taille de 64 Ko. La solution serait alors d'abandonner le microcontrôleur 8 bits doté d'un bus d'adresses 16 bits, pour doter la nouvelle version d'un microcontrôleur 16 bits capable de gérer un espace d'adresses d'au moins 1 Mo. Un tel changement entraîne une révision complète du projet. Il faut reprendre toute la conception matérielle et logicielle du produit et en plus prévoir un nouvel équipement de développement !

Si certaines familles de microcontrôleurs se sont récemment enrichies de versions équipées d'une unité de gestion de la mémoire (MMU), ce n'est, hélas, pas encore le cas de la famille 8051.

Il est relativement simple, en envisageant le sacrifice de quatre lignes de port d'E/S en plus des ports P0 et P2 déjà mobilisés pour le bus d'adresses de 16 bits, d'augmenter le champ d'adresses du 8051.

Le schéma de la figure 14.3 propose une mise en œuvre de ce principe. Tant que le bit P2.7 reste à 0, les lignes d'adresses A15, A16, A17 et A18 de L'EPROM sont maintenues à 0. On exploite alors le code programme logé dans la PAGE 0 de 32 Ko (de 0 à 7FFFH). Selon la valeur des quatre bits du port P0 (P1.0 à P1.3), lorsque P2.7 est mis à 1, on accède à une des 16 pages de 32 Ko placées en superposition les unes par rapport aux autres (overlay). Vue du processeur, la mémoire programme est constituée d'un segment classique de 0 à 7FFFH et de 16 segments superposés de 8000H à FFFFH. Un seul de ces segments n'est accessible à la fois et donc un module de programme ne doit pas chevaucher deux pages en recouvrement, car un accès à un segment paginé doit être précédé d'une commutation de page et cette commutation de page doit être assurée par le logiciel.

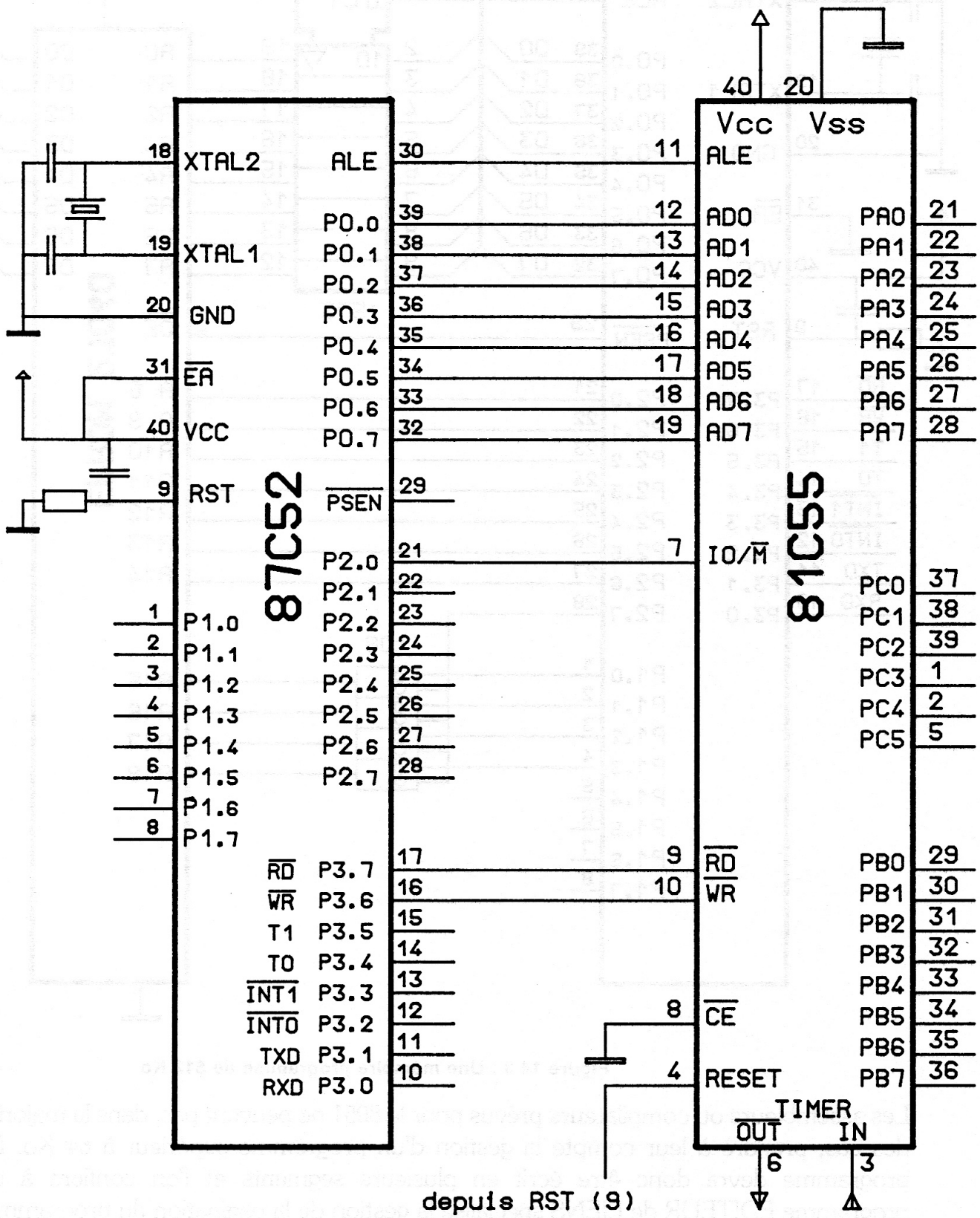


Figure 14.2 : Une commande de système en deux circuits intégrés

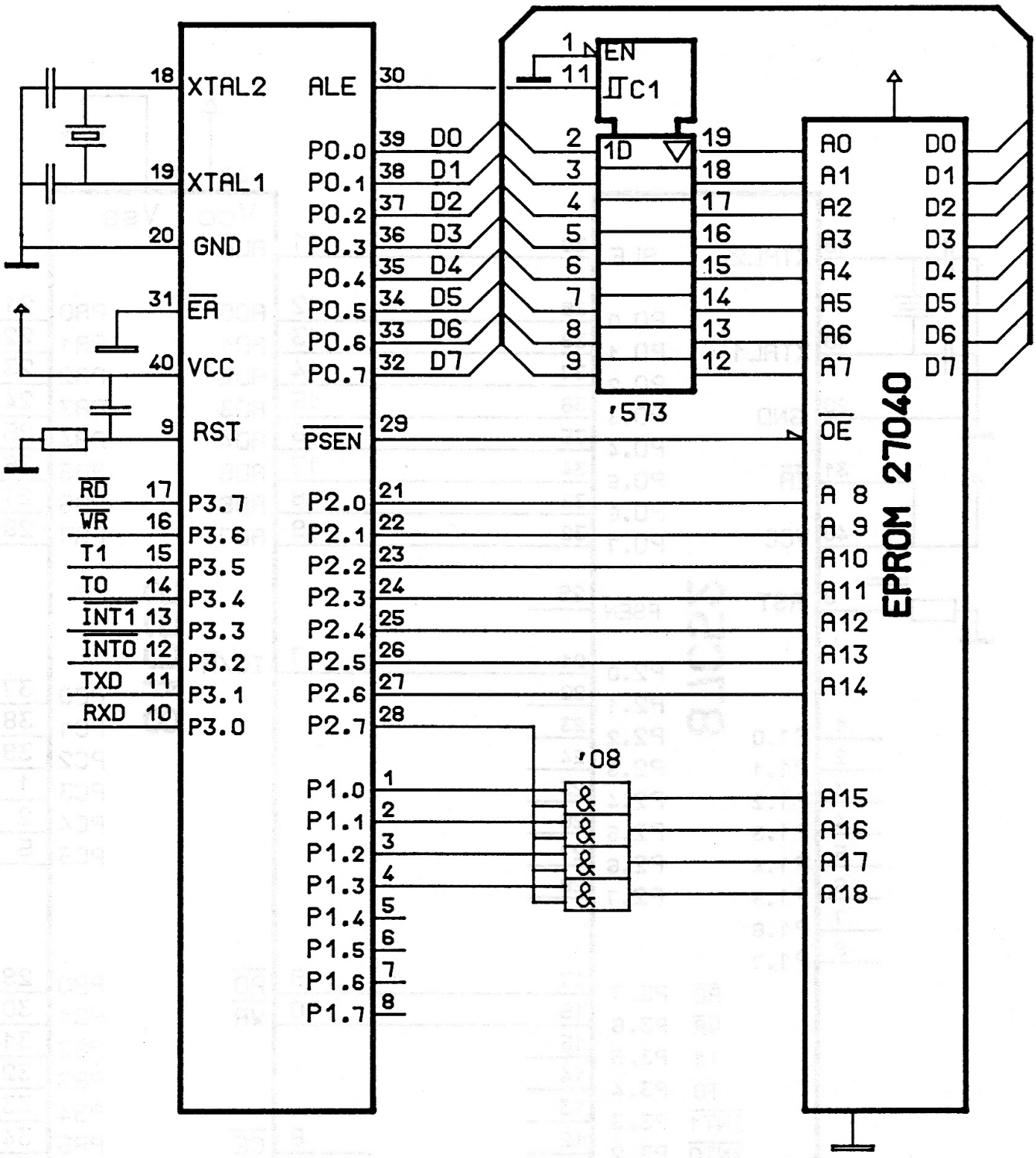


Figure 14.3 : Une mémoire programme de 512 Ko

Les assembleurs ou compilateurs prévus pour le 8051 ne peuvent pas, dans la majorité des cas, prendre à leur compte la gestion d'un programme supérieur à 64 Ko. Le programme devra donc être écrit en plusieurs segments et l'on confiera à un programme EDITEUR de LIENS spécialisé la gestion de la pagination du programme. La société RAISONANCE propose à cet effet un éditeur de liens étendu (LX-51) capable de gérer jusqu'à 31 pages de 32 Ko. Il est possible ainsi d'atteindre 1 Mo de mémoire programme.

Développement du logiciel

Après analyse et conception des traitements à effectuer, le développement d'un programme comporte trois phases :

— écriture du fichier source ;
— génération du code exécutable ;
— tests et mise au point.

Il y a quelques années, la réalisation de ces différentes opérations nécessitait des outils matériels et logiciels forts coûteux. En considérant la démocratisation des prix des programmes capables de produire le code objet exécutable, il est déconseillé d'établir "à la main" cette liste de valeurs, n'ayant de sens que pour le microcontrôleur ! Le risque d'erreurs est trop important et représente une dépense de temps et d'énergie peu justifiée. La plupart des programmes ASSEMBLEUR existent à l'heure actuelle dans des versions capables de fonctionner sur des micro-ordinateurs dits "compatibles PC". Chaque ASSEMBLEUR possède quelques particularités d'utilisation. Les exemples des chapitres précédents utilisaient des DIRECTIVES d'assemblage du programme ASM51 du constructeur INTEL. Les particularités d'un autre assembleur proposé par la société RAISONANCE seront évoquées dans les paragraphes suivants. Il faut signaler que cette société propose un ensemble EDITEUR/ASSEMBLEUR (EMA-51) à un prix "spécial enseignement" fort intéressant.

Fichier SOURCE

A l'aide d'un programme "traitement de texte" ou d'un EDITEUR, le programme est rédigé sous forme d'une suite d'instructions reconnues par le processeur (chapitre 7). Pour plus de lisibilité les instructions sont écrites sous leur forme mnémonique.

C'est le programme dit "assembleur" qui se chargera de remplacer cette représentation de l'instruction par son code binaire.

Afin de simplifier le travail de rédaction, des "pseudo-instructions" ou DIRECTIVES pour l'assembleur sont utilisables. Il est possible aussi de remplacer des valeurs numériques, peu évocatrices à leur lecture, par des symboles. Il est bien sûr conseillé de choisir des symboles ayant un sens pour améliorer la "lisibilité" du programme.

Toute cette démarche a pour but d'établir un document le plus clair possible afin de limiter les risques d'erreurs, surtout lorsque le programme commence à prendre de l'ampleur.

Il faut remarquer que certains ASSEMBLEUR ne reconnaissent pas naturellement les symboles des registres à fonction spéciale.

exemple : CLR 98H ; mise à 0 de l'indicateur RI

peut s'écrire

CLR RI

à condition de définir RI comme étant égal à 98H. On parlera alors de DECLARATIONS.

La suite de déclarations correspondant aux registres à fonction spéciale étant un problème commun à tous les développements sur un même type de microcontrôleur, les concepteurs de programmes "assembleur" ont prévu une directive autorisant l'inclusion d'une liste de déclarations :

$INCLUDE pour ASM-51 INTEL

INCLUDE pour EMA-51 RAISONANCE

Ainsi :

$NOMOD51

supprime la reconnaissance implicite des SFR du 8051 et

$INCLUDE (REG52.PDF)

permet de récupérer dans le fichier REG52.PDF les nouvelles déclarations propres au 8052.

Cette directive peut être utilisée pour lier au fichier source principal d'autres fichiers comportant des déclarations propres à l'application, ou même, des segments de programme. L'assembleur considère l'ensemble des fichiers inclus comme faisant partie du fichier principal. Il faut quand même se méfier du temps d'assemblage qui pourrait résulter d'un trop grand nombre de fichiers liés ainsi les uns aux autres !

Programmes de moyenne importance

Dans le cas d'un projet de moyenne importance le programme peut être développé sous la forme d'un seul fichier source. La directive INCLUDE peut être simplement utilisée pour la liste des registres à fonction spéciale. Dans ce cas de figure, les phases de développement se résument à :

— l'édition du fichier SOURCE ;

— l'assemblage de ce fichier.

Il résulte de cet assemblage, un fichier de codes exécutables qui peut très facilement être converti pour être placé dans l'EPROM du système. On peut considérer que cette procédure est satisfaisante pour des programmes dont le fichier source représente un document ne dépassant pas une trentaine de pages.

Développement de programmes plus conséquents

Si le développement s'avère plus complexe, il est possible d'envisager un découpage du programme en plusieurs fichiers. Les segments du programme doivent alors être "relogeables".

Une méthode consiste à développer, au sein d'un fichier principal, le corps du programme et de placer dans d'autres fichiers, des sous-programmes ou des modules utilisés par le corps principal. La directive "INCLUDE" n'est pas employée pour lier les différents fichiers. On lui préfère alors les directives PUBLIC et EXTERN pour que l'assembleur tolère l'absence apparente (dans le fichier principal) des sous-programmes et modules placés dans d'autres fichiers.

exemple :

On place dans un fichier baptisé BIBLIO un module assurant l'affichage sur un écran LCD. Ce module commence par un symbole (d'adresse relative) déclaré comme PUBLIC

```
PUBLIC AFFICHE : ....

        suite d'instructions
```

Par son attribut, le symbole AFFICHE devient alors "visible" par d'autres fichiers lors d'une édition de liens.

Par contre dans le fichier principal qui appelle par l'instruction "LCALL AFFICHE ", le module situé dans le fichier BIBLIO, le symbole AFFICHE aura été déclaré au préalable comme EXTERN. Dans ce cas, l'assembleur génère des suites de codes dites "relogeables" et l'utilisation d'un autre programme, l'ÉDITEUR de LIENS va se charger de gérer les références entre fichiers.

L'éditeur de liens a pour fonction de regrouper les fichiers de modules relogeables qui ont été assemblés séparément afin de produire finalement un fichier unique, directement utilisable pour la programmation d'une EPROM.

La démarche complète du développement se résume de la façon suivante :

— Éditions des fichiers sources ;

— Assemblage des différents modules ;

— Édition de liens pour générer le code absolu.

Pour plus de détails sur ces méthodes de travail, il est vivement conseillé de consulter la "documentation constructeur" livrée avec les programmes "assembleur" et "éditeur de liens".

A titre d'information les tableaux 14.5 et 14.6 présentent quelques directives d'assemblage utilisées dans les exemples de programmes présentés dans ce livre.

Outils de développement

Dans les documentations des outils de développement on utilise souvent les termes "système hôte" et "cible".

L'ordinateur sur lequel on développe le programme constitue le système hôte. La cible correspond au support matériel de l'application en cours de développement. La cible peut être le circuit imprimé définitif de l'application ou une carte sur laquelle les composants sont interconnectés par une technique de câblage nommée "Wrapping". Sur la cible on trouve donc un microcontrôleur accompagné ou non d'une mémoire programme (EPROM).

Lorsque le programme est écrit et que l'assemblage, accompagné ou non d'une édition de liens, a permis d'obtenir un fichier de codes exécutables, ces codes sont encore dans le système hôte. En reliant au système hôte un "programmateur d'EPROM", il est possible de transférer le fichier dans une EPROM. L'EPROM est ensuite placée sur la cible et on procède au premier test. L'essai n'est pas concluant : que doit-on mettre en cause, le programme, le câblage ou les deux ! Pour lever les doutes sur la partie "matériel" il est possible de prévoir un petit programme de tests. Mais il faudra, à chaque nouvel essai, effacer l'EPROM et la programmer à nouveau. La suite de ces opérations prend du temps et une EPROM n'est pas "reprogrammable" indéfiniment.

Simulateur de programmes

Pour lever les soupçons sur la partie "programme", l'idéal serait de pouvoir demander au microprocesseur du système hôte de simuler le déroulement du programme. Ce genre de programme existe sur le marché. Il permet de suivre sur l'écran de l'ordinateur le déroulement du programme, ses actions sur les registres, indicateurs et zones mémoire de données. Il est même possible de simuler des événements externes tels que communication sur le port série, mesure de tension pour les microcontrôleurs équipés de convertisseur analogique numérique, etc...

Il devient possible de corriger un grand nombre d'erreurs avant de passer à un essai sur la cible.

En plus des services rendus, ce genre de programme possède des qualités didactiques indéniables. L'acquisition d'un tel outil est indispensable pour un bureau d'étude ou un centre de formation.

Simulateur d'EPROM

Le simulateur se substitue à l'EPROM placée sur la carte cible. C'est en fait une mémoire RAM téléchargeable depuis le système hôte. L'avantage de ce simulateur est de pouvoir tester et modifier rapidement un programme en supprimant toute manipulation de composant. Son faible coût et le gain de temps qu'il apporte en font un outil indispensable. Une fois l'application au point, le programme sera placé dans une véritable EPROM et prendra place définitivement sur la carte cible.

Émulateurs

Un émulateur remplace le microcontrôleur sur la carte cible. Sa conception, beaucoup plus complexe qu'un simulateur d'EPROM, en fait un produit généralement beaucoup plus cher. L'émulateur exécute les instructions, produit les signaux comme son modèle et ce dans un temps pratiquement identique. Mais en plus il est capable de mémoriser des "traces" des événements et peut aussi fonctionner en mode pas à pas. Cet outil permet aussi bien une mise au point du logiciel qu'un dépistage efficace des problèmes dus au matériel. Son utilisation est recommandée dans les applications délicates où la mise au point du programme est très liée au support matériel du projet.

ASM-51 FONCTION

CSEG	AT adr	définit le début d'un segment de code exécutable à partir de l'adresse absolue adr
DSEG	AT adr	définit le début absolu d'une zone RAM interne
XSEG	AT adr	définit le début absolu d'une zone RAM externe
BSEG	AT adr	définit le début d'une zone adressable au niveau du bit.
EQU		affecte une valeur numérique à un symbole
DB		Insère une suite de valeurs au format "octet"
DS	n	réserve n octets
DW		insère une suite de valeurs de format 16 bits.
DBIT	n	réserve et déplace compteur de bit adressable.
END		fin du programme.

Exemple de mise en œuvre :

```
NBR_FOIS        EQU 5          ;le symbole correspond à la valeur 5.
        DSEG    AT 30H         ;Dans la RAM interne à partir de l'adresse
MESURE :        DS  1          ;30H on réserve 1 octet pour la variable
                               ;MESURE puis à la suite (31 et 32H)
CALCUL :        DS  2          ;on réserve 2 octets pour CALCUL.

        BSEG    AT  0          ;A partir de l'adresse 20H le premier bit
FLAG_IT:        DBIT 1         ;d'adresse 0 est réservée à FLAG_IT
BASCULE :       DBIT 1          ;puis le bit suivant est réservé à BASCULE

        XSEG    AT  0          ;Dans la RAM externe à partir de 0
TAMPON :        DS  32         ;on réserve 32 octets pour TAMPON
CONFIG :        DS  5          ;puis à la suite 5 octets pour CONFIG

        CSEG    AT  0          ;segment de code absolu à l'adresse 0
        LJMP    DEPART          ;saut vers adresse DEPART après un RESET
                               ;cette instruction occupe 3 octets.
                               ;pas de code placé avant l'adresse 23H
        CSEG    AT  23H
SERIE :                         ;sous-programme d'interruption
        suite d'instructions
        ..............
        ..............
DEPART :                        ;début du traitement principal
        ..............
        ..............
        END                     ;fin du fichier
```

Tableau 14.5 : Quelques directives pour ASSEMBLEUR ASM51

EMA-51		FONCTION
CODE		spécifie un segment de codes relogeables
CODE	AT adr	définit le début absolu d'un segment de codes
DATA		spécifie un segment en RAM interne relogeable
DATA	AT adr	définit le début absolu d'un segment de RAM interne
XDATA		spécifie un segment de données en RAM externe
XDATA	AT adr	définit le début absolu d'une zone RAM externe
BIT		spécifie un segment adressable au niveau du bit
BIT	AT adr	définit le début absolu d'une zone "BIT"
NUMBER		affecte une valeur numérique à un symbole
DB	n	réserve n octets
DB	=	Insère une suite de valeurs au format "octet"

Exemple :

```
NB_FOIS         NUMBER 5        ;le symbole correspond à la valeur 5 .

                DATA    AT 30H  ;Dans la RAM interne à partir de l'adresse
MESURE:         DB  1           ;30H on réserve 1 octet pour la variable
                                ;MESURE puis à la suite (31 et 32H)
CALCUL:         DB  2           ;on réserve 2 octets pour CALCUL.

                BIT     AT  0   ;A partir de l'adresse 20H le premier bit

FLAG_IT :       DB  1           ;d'adresse 0 est réservée à FLAG IT
BASCULE:        DB  1           ;puis le bit suivant est réservé à BASCULE

                XDATA   AT  0   ;Dans la RAM externe à partir de 0
TAMPON:         DB  32          ;on réserve 32 octets pour TAMPON
CONFIG:         DB  5           ;puis à la suite 5 octets pour CONFIG
                CODE    AT  0   ;segment de code absolu à l'adresse 0
                LJMP    DEPART  ;saut vers adresse DEPART après un RESET
                                ;cette instruction occupe 3 octets.
                                ;pas de code placé avant l'adresse 23H
                CODE    AT  23H
SERIE:                          ;sous-programme d'interruption
                suite d'instructions
                ..............
                ..............
DEPART:                         ;début du traitement principal
                ..............
                ..............
DB              = "SALUT"       ;insère les codes ASCII de la chaîne SALUT
```

Tableau 14.6 : Quelques directives pour ASSEMBLEUR EMA-51

15

Ressources logicielles

Module arithmétique

Les sous-programmes de ce paragraphe, inspirés d'une note l'application de Rick Schue publiée par Intel Corporation, permettent de réaliser des opérations arithmétiques sur des variables d'au moins 16 bits. Ils utilisent tous une même zone mémoire de 17 octets qui doit être située en RAM interne adressable aussi bien en mode direct qu'en mode indirect par registre. Dans l'exemple de déclaration de cette zone mémoire, l'adresse 30H a été retenue. Les variables REG_A et REG_B représentent deux registres "opérandes", et ACCU, le registre de 32 bits dans lequel le résultat de l'opération est rangé. Le registre REMAIN représente le reste d'une division, il doit être de 24 bits pour satisfaire les besoins de l'algorithme de la division 32 bits par 16 bits. Le registre BCD reçoit les trois octets d'une conversion en code BCD d'un entier de 16 bits. La conversion de la valeur FFFFH place dans ce registre les valeurs suivantes :

BCD+2	BCD+1	BCD
06	55	35

Avant d'appeler une de ces fonctions de calcul, il suffit de pointer avec R0 et R1 les deux variables à traiter. Le sous-programme se charge alors de placer dans les registres REG_A et REG_B les valeurs des deux variables, puis effectue le calcul, et replace le pointeur R0 sur le résultat (poids faible de ACCU).

Un petit programme est donné comme exemple d'utilisation de ces sous-programmes :

17 octets réservés a partir de 30

DSEGAT (30H)

```
;
DIVCTR :    DS      1       ; compteur pour division
REG_A :     DS      4       ; registre A 16 bits ou 32 bits pour div. 32/16
REMAIN :    DS      3       ; reste de la division (extension de A)
REG_B :     DS      2       ; registre B 16 bits
ACCU :      DS      4       ; accumulateur 32 bits
BCD :       DS      3       ; registre résultat de la conversion BCD
```

Exemple de programme utilisant le module arithmétique
Le résultat d'une conversion analogique numérique 0/5 volts sur 10 bits
représente une tension entre 0 et 400 volts

```
;variables spécifiques à l'exemple
MESURE :    DS      2       ; résultat mesure faite par un C.A.N
CALIBRE :   DS      2       ;
ECHELLE :   DS      2
;POINT D'ENTREE APRES RESET
            CSEG    AT 0000H    ; ADRESSE du vecteur RESET
            ORG     0H
            JMP     DEBUT
ORG         100H
DEBUT :     MOV     DPTR,#400   ; calibre 400 volts par exemple
            MOV     CALIBRE,DPL ; mis en place en mémoire
            MOV     CALIBRE+1,DPH
            MOV     DPTR,#1024  ; Echelle de 1024 pour 10 bits
            MOV     ECHELLE,DPL
            MOV     ECHELLE+1,DPH
;Après cette initialisation faire le calcul
            MOV     R1,#CALIBRE ; R1 pointe le calibre
            MOV     R0,#MESURE  ; R0 pointe la mesure
            LCALL   MUL_AB      ; faire MESURE x CALIBRE
            MOV     R1,#ECHELLE ; R1 pointe la valeur de l'échelle
```

```
        LCALL   DIV__TS     ; Faire (MESURExCALIBRE)/ECHELLE
        LCALL   HEX__BCD    ; convertir en code BCD
```

Les valeurs de BCD+1 et de BCD peuvent alors être affichées
BCD représente dizaine et unité du résultat et les quatre bits de poids faible de BCD+1 représentent le chiffre des centaines de volts.

Chargement dans REG__A d'un 16 bits pointé par R0

```
LOAD__16 :  MOV     REG__A+0,@R0
            INC     R0
            MOV     REG__A+1,@R0
            MOV     REG__A+2,#0
            MOV     REG__A+3,#0
            RET
```

Chargement dans REG__A d'un 32 bits pointé par R0

```
LOAD__32 :  MOV     REG__A+0,@R0
            INC     R0
            MOV     REG__A+1,@R0
            INC     R0
            MOV     REG__A+2,@R0
            INC     R0
            MOV     REG__A+3,@R0
            RET
```

Multiplication 16 bits par 16 bits

L'algorithme utilisé est calqué sur la méthode que l'on utilise lorsqu'on effectue une multiplication "à la main". En admettant que A, B, C et D représentent des octets, AB et CD représentent des valeurs de 16 bits :

soit l'opération AB x CD

On effectue D x B puis D x A, ensuite C x B et C x A.

Il est très facile de multiplier deux octets entre eux grâce à l'instruction MUL AB. Seule la gestion des résultats et des sommes avec décalage demande un peu d'attention.

MULTIPLICATION 16 X 16 bits

entrée : R0 et R1 pointe le poids faible des deux variables

sortie : résultat dans ACCU pointé par R0

```
MUL_AB :    ACALL   LOAD_16         ;charger REG_A avec variable (R0)
            MOV     REG_B,@R1       ;placer variable pointée par R1
            INC     R1              ;dans REG_B
            MOV     REG_B+1,@R1
            ACALL   AB_MUL          ; effectuer multiplication
            MOV     R0,#ACCU        ; poids faible résultat pointé par R0
            RET
;
AB_MUL :    MOV     ACCU+2,#0       ; poids fort accumulateur à 0
            MOV     ACCU+3,#0       ;
            MOV     A,REG_A         ; multiplier les poids faibles
            MOV     B,REG_B         ; ( idem D x B)
            MUL     AB
            MOV     ACCU,A          ; résultat provisoire dans ACCU
            MOV     ACCU+1,B
            MOV     A,REG_A+1       ; poids fort A par poids faible B
            MOV     B,REG_B         ; (idem D x A)
            MUL     AB
            ADD     A,ACCU+1        ; somme avec partiel précédent
            MOV     ACCU+1,A        ; avec un décalage vers gauche
            MOV     A,B
            ADDC    A,ACCU+2        ; prise en compte d'un report
            MOV     ACCU+2,A        ; vers ACCU+2
            JNC     AB_M1           ;
            INC     ACCU+3          ; et vers ACCU+3
AB_M1 :     MOV     A,REG_A         ; multiplier poids faible A par
            MOV     B,REG_B+1       ; poids fort de B
            MUL     AB              ; (idem C x B)
            ADD     A,ACCU+1
            MOV     ACCU+1,A
            MOV     A,B
            ADDC    A,ACCU+2
```

```
                MOV     ACCU+2,A
                JNC     AB__M2
                INC     ACCU+3
AB__M2 :        MOV     A,REG__A+1      ; multiplier poids fort de A
                MOV     B,REG__B+1      ; avec poids fort de B
                MUL     AB              ; ( idem A x B)
                ADD     A,ACCU+2        ; somme avec décalage
                MOV     ACCU+2,A
                MOV     A,B
                ADDC    A,ACCU+3        ; en tenant compte des reports
                MOV     ACCU+3,A
                RET
;
```

Division 32 bits par 16 bits

Le résultat de cette division est situé dans ACCU sur 32 bits. L'exécution de ce sous-programme nécessite entre 1700 et 1960 cycles machine. Le temps d'exécution pourrait être considérablement réduit en admettant que le quotient soit limité à 16 bits, mais il faut alors se méfier des risques de débordement.

---

diviser (R0) (32 bits) par (R1) (16bits)
sortie :
quotient dans ACCU (32bits) pointé par R0
et (R1) reste sur 16 bits

```
DIV__TS :       ACALL   LOAD__32        ; (R0) placé dans REG__A
                MOV     REG__B,@R1
                INC     R1
                MOV     REG__B+1,@R1
                ACALL   DIV__32
                MOV     R0,#ACCU        ; quotient pointé par R0
                MOV     R1,#REMAIN      ; reste pointé par R1
                RET
;
DIV__32 :       MOV     REMAIN,#0       ;reste à 0
                MOV     REMAIN+1,#0
                MOV     REMAIN+2,#0
```

```
                MOV     DIVCTR,#32      ; reprendre DIV__T1 32 fois
;
DIV__T1 :       ACALL   SH__DIVI        ; décalage gauche du dividende
                CLR     C               ; tester si reste > diviseur
                MOV     A,REMAIN
                SUBB    A,REG__B
                MOV     A,REMAIN+1
                SUBB    A,REG__B+1
                MOV     A,REMAIN+2
                SUBB    A,#0
                JNC     SUB__T          ;reste > diviseur alors soustraire
                SJMP    QUOT__T         ;sinon introduire un 0 dans résultat
SUB__T:         CLR     C
                MOV     A,REMAIN        ;soustraire diviseur du reste partiel
                SUBB    A,REG__B
                MOV     REMAIN,A
                MOV     A,REMAIN+1
                SUBB    A,REG__B+1
                MOV     REMAIN+1,A      ;
                MOV     A,REMAIN+2
                SUBB    A,#0
                MOV     REMAIN+2,A      ;
;
QUOT__T :       CPL     C               ; introduire report dans
                MOV     A,ACCU          ; quotient
                RLC     A               ; 0 si reste < diviseur
                MOV     ACCU,A          ; 1 si reste > diviseur
                MOV     A,ACCU+1
                RLC     A
                MOV     ACCU+1,A
                MOV     A,ACCU+2
                RLC     A
                MOV     ACCU+2,A
                MOV     A,ACCU+3
                RLC     A
                MOV     ACCU+3,A
                DJNZ    DIVCTR,DIV__T1  ;
                RET
;
```

```
SH__DIVI :  CLR C
            MOV   A,REG__A
            RLC   A
            MOV   REG__A,A
            MOV   A,REG__A+1
            RLC   A
            MOV   REG__A+1,A
            MOV   A,REG__A+2
            RLC   A
            MOV   REG__A+2,A
            MOV   A,REG__A+3
            RLC   A
            MOV   REG__A+3,A     ; bit poids fort dans Cy
            MOV   A,REMAIN       ; placer report dans poids faible
            RLC   A              ; du reste partiel
            MOV   REMAIN,A
            MOV   A,REMAIN+1     ; sur 16 bits
            RLC   A
            MOV   REMAIN+1,A
            MOV   A,REMAIN+2
            RLC   A
            MOV   REMAIN+2,A
            RET
```

Division 16 bits par 16 bits

DIVISION 16/16
diviser (R0) (16 bits) par (R1) (16bits)
sortie :
quotient dans ACCU (16 bits) pointé par R0
et (R1) reste sur 16 bits

```
DIV__SS :   ACALL  LOAD__16      ; (R0) placé dans REG__A
            MOV    REG__B,@R1
            INC    R1
            MOV    REG__B+1,@R1
            ACALL  DIV__32
            MOV    R0,#ACCU      ; quotient pointé par R0
```

```
                MOV     R1,#REMAIN      ; reste pointé par R1
                RET
DIV__16 :       MOV     REMAIN,#0       ; reste à 0
                MOV     REMAIN+1,#0
                MOV     REMAIN+2,#0
                MOV     DIVCTR,#16      ; compteur pour opération
DIV__S1 :       ACALL   SH__DIVI__S     ; décalage gauche du dividende
                CLR     C               ; tester si reste > diviseur
                MOV     A,REMAIN
                SUBB    A,REG__B
                MOV     A,REMAIN+1
                SUBB    A,REG__B+1
                MOV     A,REMAIN+2
                SUBB    A,#0
                JNC     SUB__S          ; reste > diviseur alors soustraire
                SJMP    QUOT__S         ; sinon introduire 0 dans résultat
SUB__S :        CLR     C
                MOV     A,REMAIN        ; soustraire diviseur du reste partiel
                SUBB    A,REG__B
                MOV     REMAIN,A
                MOV     A,REMAIN+1
                SUBB    A,REG__B+1
                MOV     REMAIN+1,A
                MOV     A,REMAIN+2
                SUBB    A,#0
                MOV     REMAIN+2,A      ;
QUOT__S :       CPL     C
                MOV     A,ACCU          ; Cy introduit dans quotient
                RLC     A
                MOV     ACCU,A
                MOV     A,ACCU+1
                RLC     A
                MOV     ACCU+1,A
                DJNZ    DIVCTR,DIV__S1
                RET
SH__DIVI__S :
                CLR     C
                MOV     A,REG__A
                RLC     A
                MOV     REG__A,A
```

```
        MOV     A,REG__A+1
        RLC     A
        MOV     REG__A+1,A
        MOV     A,REMAIN        ;placer report dans poids faible
        RLC     A               ; du reste partiel
        MOV     REMAIN,A
        MOV     A,REMAIN+1      ; sur 16 bits
        RLC     A
        MOV     REMAIN+1,A
        MOV     A,REMAIN+2
        RLC     A
        MOV     REMAIN+2,A
        RET
```

Conversion en format BCD

---

convertir en BCD le 16 bits pointé par R0
R0 pointe l'octet de poids faible des 16 bits
la conversion est rangée dans BCD+2 (poids fort)
BCD+1
BCD+0 (poids faible)

```
HEX__BCD :
        ACALL   LOAD__16
        MOV     REG__B,#10H
        MOV     REG__B+1,#27H   ; division par 10000
        ACALL   DIV__16
        MOV     BCD+2,ACCU      ; poids fort dans BCD+2
        MOV     R0,#REMAIN      ; reste pointé par R0
        ACALL   LOAD__16        ; charger reste dans REG__A
        MOV     REG__B,#0E8H    ; division par 1000
        MOV     REG__B+1,#3
        ACALL   DIV__16
        MOV     A,ACCU
        SWAP    A
        MOV     BCD+1,A
```

```
        MOV     R0,#REMAIN      ; reste pointé par R1
        ACALL   LOAD_16         ; charger reste dans REG_A
        MOV     REG_B,#64H      ; division par 100
        MOV     REG_B+1,#0
        ACALL   DIV_16
        MOV     A,ACCU
        ORL     BCD+1,A
        MOV     A,REMAIN        ; reste inférieur à 100
        MOV     B,#10           ; division par 10
        DIV     AB
        SWAP    A               ; A = dizaine
        ORL     A,B             ; placer unite
        MOV     BCD,A           ; rangement dizaine et unité
        RET
```

Division par 256

DIVISION DE L'ACCUMULATEUR de 32 bits par 256
Reste dans REMAIN (1 octet)

```
DIV_256 :   MOV     REMAIN,ACCU
            MOV     ACCU,ACCU+1
            MOV     ACCU+1,ACCU+2
            MOV     ACCU+2,ACCU+3
            MOV     ACCU+3,#0
            RET                     ; 12 cycles machine
```

Gestion d'un afficheur L.C.D

Le module de gestion proposé permet de commander un afficheur à cristaux liquides de deux lignes de 16 caractères. L'analyse de nombreuses documentations laisse apparaître une grande similitude de conception des différents modèles proposés par divers constructeurs. Seules la disposition des connections et la tension de polarisation nécessaire au réglage de contraste peuvent changer d'une marque à l'autre.

Les sous-programmes proposés permettent aussi de commander d'autres modèles d'afficheurs comportant un nombre de lignes (ou un nombre de caractères par ligne) différent du modèle pris comme exemple.

Chaque caractère possède une adresse. Les adresses des caractères de la deuxième ligne ne sont pas consécutives aux adresses des caractères de la première ligne. Ainsi, quelle que soit la longueur de ligne du modèle choisi (16, 20, 24 ou 40 caractères), l'adresse du premier caractère de la deuxième ligne est toujours 40H. Ces adresses permettent de positionner directement le curseur sur une zone déterminée de l'afficheur. Le code de contrôle à envoyer est composé de l'adresse (sur 7 bits) précédé d'un bit à 1. Dans le programme ci-après l'adresse 40H se traduit donc par le code de contrôle C0H.

Le schéma de la figure 15.1 propose une solution d'interfaçage entre le microcontrôleur et l'afficheur. Le décodeur d'adresses détermine des "pages" de 8 Ko dans l'espace de mémoire de données. La première page (de 0 à 1FFFH) est laissée libre pour une éventuelle mémoire RAM du type 6264. L'afficheur bloque les 8 Ko de la page suivante (de 2000H à 3FFFH), alors qu'un espace de quatre adresses suffit à sa commande. Cette solution de décodage peut être modifiée et améliorée mais il est important de maintenir les lignes A0 et A1 respectivement sur les entrées R/W et RS de l'afficheur.

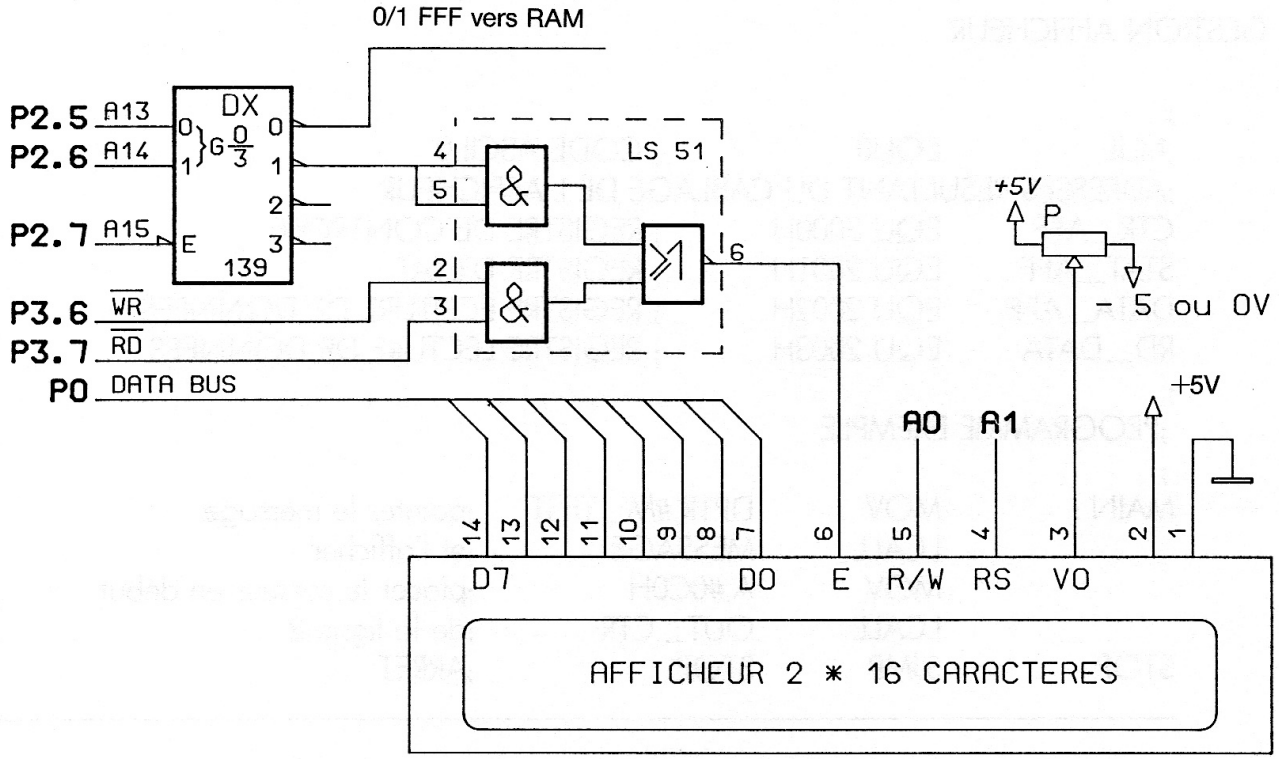


Figure 15.1 : Schéma de câblage de l'afficheur

Le tableau 15.2 précise le rôle de ces entrées de contrôle.

RS	R/W	fonction
A1	A0	
0	0	écriture dans le registre de contrôle
0	1	lecture du registre d'état (indicateur BUSY)
1	0	écriture d'un caractère (registre données)
1	1	lecture du registre de données

Note sur le sous-programme INI_AFF

La procédure d'initialisation de l'afficheur n'est garantie que si la mise sous tension de l'afficheur s'effectue dans le respect de la condition suivante : la tension entre les bornes 1 et 2 doit atteindre 5 volts dans un laps de temps compris entre 0.1 ms et 10 ms maximum. Dans le cas contraire l'initialisation du microcontrôleur interne à l'afficheur risque de ne pas être assurée correctement. Une procédure logicielle particulière doit alors forcer cette initialisation. Pour plus ample détail, il est conseillé de consulter la documentation constructeur du modèle d'afficheur utilisé.

GESTION AFFICHEUR

```
;
;NUL            EQU0              ; CODE ASCII 0
;ADRESSES RESULTANT DU CABLAGE DE L'AFFICHEUR
CTR_AFF        EQU 2000H         ; REGISTRE DE CONTROLE
STAT_AFF       EQU 2001H         ; REGISTRE D'ETAT
DATA_AFF       EQU 2002H         ; REGISTRE ECRITURE DE DONNEES
RD_DATA        EQU 2003H         ; REGISTRE LECTURE DE DONNEES
;
 ;PROGRAMME EXEMPLE
 ;
MAIN :         MOV      DPTR,#M_TEST    ;pointer le message
               LCALL    MESSAGE         ;et l'afficher
               MOV      A,#0C0H         ;placer le curseur en début
               LCALL    OUT_CTR         ;de la ligne 2
STOP :         SJMP     STOP            ;ARRET
```

ENVOIE VERS AFFICHEUR LA CHAINE en ROM
pointée PAR DPTR
FIN DE CHAINE SIGNALEE PAR "NUL"

```
MESSAGE :   CLR     A           ; récupérer donnée
            MOVC    A,@A+DPTR   ; pointée par DPTR
MES1 :      CJNE    A,#NUL,MES  ; terminaison ?
            RET                 ; oui alors fin
MES :       ACALL   OUT_CHAR    ; sinon envoi
            INC     DPTR        ; et refaire
            SJMP    MESSAGE
```

AFFICHEUR CURSEUR CLIGNOTANT

```
CURS_ON :   PUSH    ACC
            MOV     A,#0DH      ;curs on + BLINK
            ACALL   OUT_CTR
            POP     ACC
            RET
```

SUPPRIME CURSEUR

```
CURS_OFF :  PUSH    ACC
            MOV     A,#0CH
            ACALL   OUT_CTR
            POP     ACC
            RET
```

ENVOI A=CODE DE CONTROLE VERS AFFICHEUR

```
OUT_CTR :   PUSH    DPH
            PUSH    DPL
            ACALL   FREE_AFF
            MOV     DPTR,#CTR_AFF
            MOVX    @DPTR,A
            POP     DPL
            POP     DPH
            RET
```

ENVOI A = CARACTERE VERS AFFICHEUR

```
OUT_CHAR :  PUSH    DPH
            PUSH    DPL
            ACALL   FREE_AFF
            MOV     DPTR,#DATA_AFF
            MOVX    @DPTR,A
            POP     DPL
            POP     DPH
            RET
```

ATTENTE AFFICHEUR LIBRE
BIT 7 → BUSY = 1 afficheur non disponible

```
FREE_AFF :  PUSH    ACC             ;sauver contexte
            PUSH    DPH
            PUSH    DPL
            MOV     DPTR,#STAT_AFF  ;pointer registre ETAT
FREE_0 :    MOVX    A,@DPTR         ;lire l'état
            JB      ACC.7,FREE_0    ;si BUSY=1 attendre
            POP     DPL             ;sinon afficheur
            POP     DPH             ;prêt à recevoir
            POP     ACC
            RET
```

INITIALISE L'AFFICHEUR

```
INI_AFF :   MOV     A,#38H          ;ACTIVE LES DEUX LIGNES
            ACALL   OUT_CTR
CLR_AF :    MOV     A,#01H          ;"HOME CURSOR"
            ACALL   OUT_CTR
            MOV     A,#0CH          ;display ON et curseur OFF
            ACALL   OUT_CTR
            MOV     A,#06           ;GAUCHE VERS DROITE
            ACALL   OUT_CTR
            RET
M_TEST :    DB      'AFFICHEUR OK !',0
```

Annexe

Précis de programmation

Mot d'état programme :

PSW adresse directe D0H adressable au niveau bit

	D7	D6	D5	D4	D3	D2	D1	D0
D0H	CY	AC	F0	RS1	RS0	OV	—	P

CY : indicateur de retenue (désigné aussi par la lettre C)
AC : indicateur de retenue auxiliaire indiquant un report du bit 3 vers le bit 4 (utilisé pour les opérations en code BCD).
F0 : indicateur F0. Sans fonction, mis à disposition de l'utilisateur.
RS1 : Sélection d'une banque de registres.
RS0 : Sélection d'une banque de registres.
OV : indicateur de débordement.
P : indicateur de parité.

RS1	RS0	banque active	adresse
0	0	0	00-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

Instructions modifiant les indicateurs

Instruction	Cy	Ov	AC	Instruction	Cy	Ov	AC
ADD	X	X	X	SETB C	1		
ADDC	X	X	X	CLR C	0		
SUBB	X	X	X	CPL C	X		
MUL	0	X		ANL C,BIT	X		
DIV	0	X		ANL C,/BIT	X		
DA	X			ORL C,BIT	X		
RRC	X			ORL C,/BIT	X		
RLC	X			MOV C,BIT	X		
CJNE	X						

TMOD : Mode de fonctionnement des TIMERS 0 et 1

adresse directe 89H

GATE	C/T	M1	M0	GATE	C/T	M1	M0

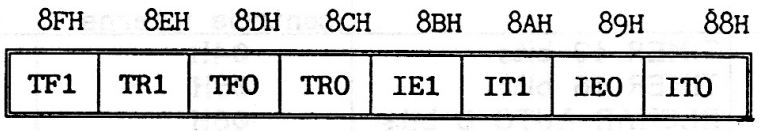
GATE : GATE = 1, le TIMER x est validé seulement lorsque sa broche d'entrée "INTx" est à l'état 1 et que le bit "TRx" du registre TCON est à 1.
GATE = 0, le TIMER est validé si "TRx" = 1.

C/T : Permet la sélection de la fonction TIMER :
C/T = 0 le TIMER est en fonction TEMPORISATEUR,
C/T = 1 le TIMER est en fonction COMPTEUR à partir des événements présents sur la broche d'entrée "Tx".

M1	M0	Mode
0	0	compteur 13 bits compatible au microcontrôleur MCS-48
0	1	compteur 16 bits
1	0	compteur 8 bits à rechargement automatique. Le contenu de THx est rechargé dans TLx lorsque celui-ci repasse à 0.
1	1	TIMER 0: Le registre TL0 devient un compteur 8 bits contrôlé normalement par les bits de contrôle du TIMER 0. Par contre, TH0 est configuré en temporisateur 8 bits contrôlé par les bits de contrôle du TIMER 1.
1	1	TIMER 1: compteur 1 à l'arrêt.

TCON : Contrôle des Timers 0 et 1

adresse directe 88H adressable au niveau bit



TF1 : Indicateur de débordement du TIMER 1. Mis à 1 automatiquement lorsque le compteur repasse à 0. Si l'interruption correspondante est autorisée, cet indicateur est remis à 0 automatiquement lorsque le sous-programme d'interruption est exécuté.

TR1 : Bit de déclenchement du TIMER 1. Il doit être positionné par logiciel à 1 (pour lancer) ou 0 (pour arrêter) le TIMER.

TF0 : Indicateur de débordement du TIMER 0. Identique à TF1.

TR0 : Bit de déclenchement du TIMER 0. même fonctionnement que TR1.

IE1 : Indicateur de transition sur la source externe INT1. Cet indicateur est mis à 1 automatiquement lorsqu'une transition est détectée sur la borne INT1. Lors du service de l'interruption cet indicateur est remis à 0.

IT1 : Sélection du type d'événement devant déclencher l'interruption depuis le source INT1.
IT1 = 0 interruption sur détection d'un niveau 0 sur INT1.
IT1 = interruption sur transition négative (1 vers 0).

IE0 : Indicateur de transition sur la source externe INT0.
Fonctionnement identique à IE1.

IT0 : IT0 = interruption sur détection d'un niveau 0 sur INT0.
IT0 = interruption sur détection d'une transition négative.

Initialisation d'un Timer

Les valeurs données dans les quatre tableaux suivants permettent d'initialiser un seul Timer à la fois. Pour initialiser les deux timers simultanément, la valeur d'initialisation s'obtient par somme logique des deux valeurs.

Timer 0 comme temporisation

MODE	Fonction	VALEUR DE TMOD	
		contrôle interne	contrôle externe
0	TIMER 13 bits	00H	08H
1	TIMER 16 bits	01H	09H
2	RECHAR.AUTO 8 bits	02H	0AH
3	2 timers 8 bits	03H	0BH

Timer 0 comme compteur

MODE	Fonction	VALEUR DE TMOD	
		contrôle interne	contrôle externe
0	TIMER 13 bits	04H	0CH
1	TIMER 16 bits	05H	0DH
2	RECHAR.AUTO 8 bits	06H	0EH
3	2 timers 8 bits	07H	0FH

Timer 1 comme temporisation

MODE	Fonction	VALEUR DE TMOD	
		contrôle interne	contrôle externe
0	TIMER 13 bits	00H	80H
1	TIMER 16 bits	10H	90H
2	RECHAR.AUTO 8 bits	20H	A0H
3	ne fonctionne pas	30H	B0H

Note :
contrôle interne : le timer est lancé ou arrêté par le bit TR0
contrôle externe : le timer est lancé ou arrêté par transition négative (1 vers 0) sur INT0 si TR0 = 1

Timer 1 comme compteur

MODE	Fonction	VALEUR DE TMOD	
		contrôle interne	contrôle externe
0	TIMER 13 bits	40H	C0H
1	TIMER 16 bits	50H	D0H
2	RECHAR.AUTO 8 bits	60H	E0H
3	indisponible	-	-

Note :
contrôle interne : le timer est lancé ou arrêté par le bit TR1
contrôle externe : le timer est lancé ou arrêté par transition négative sur INT1 si TR1 = 1

Contrôle du TIMER 2 (sur 8052 seulement)

T2CON **adresse directe C8H** **adressable au niveau bit**

CFH	CEH	CDH	CCH	CBH	CAH	C9H	C8H
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$

TF2: Dépassement de capacité. Mis à 1 lorsque le Timer 2 passe de la valeur FFFFH à la valeur 0.

EXF2: Indicateur externe. Mis à 1 lorsqu'une capture ou un rechargement est provoqué par transition 0 vers 1 sur broche T2EX et que EXEN2 = 1. Si l'interruption du Timer 2 est autorisée et que EXF2=1 alors le sous-programme placé à l'adresse 2BH est exécuté. La remise à 0 de EXF2 doit être assurée par programmation.

RCLK: Bit de contrôle de l'horloge de réception. RCLK =1 positionné à 1, le port série, programmé en mode 1 ou 3, utilise l'impulsion de dépassement du Timer 2 comme signal d'horloge de réception. RCLK =0, c'est l'impulsion de dépassement du Timer 1 qui joue ce rôle.

TCLK: Contrôle de l'horloge d'émission. TCLK = 1, le port série, (en mode 1 ou 3), utilise l'impulsion de dépassement du Timer 2 comme signal d'horloge d'émission. TCLK = 0, c'est l'impulsion de dépassement du Timer 1 qui joue ce rôle.

EXEN2: Validation du mode externe. EXEN2 = 1: autorisation de capture ou de rechargement provoqué par une transition négative détectée sur broche T2EX. Ceci n'est possible que si le Timer 2 n'est pas utilisé comme horloge du port série (TCLK et RCLK à 0). Le fait de forcer EXEN2 à 0 permet d'ignorer les événements présents sur la broche T2EX.

TR2 : Contrôle du fonctionnement. La mise à 1 de TR2 provoque le démarrage du Timer.

C/$\overline{T2}$: Sélection du mode de fonctionnement du Timer 2:
C/T2=0 fonction "temporisateur";
C/T2=1 fonction "compteur d'événements externes".
En mode temporisateur, c'est l'horloge (du microcontrôleur) divisée par 12 qui commande l'incrémentation du Timer. En mode compteur, seules des transitions négatives peuvent être détectées comme évènements externes.

CP/$\overline{RL2}$: sélection pour capture ou rechargement. CP/RL2 = 1 le mode capture est sélectionné. La conjugaison EXEN2=1 et détection d'une transition négative sur T2EX permet une opération de capture: la valeur contenue par le registre 16 bits du Timer 2 est placée dans les registres RCAP2H et RCAP2L. Ce mode de capture n'est possible que si RCLK et TCLK sont à 0. Dans le cas contraire, c'est le mode rechargement automatique qui est sélectionné.

INITIALISATION DU TIMER 2

exceptés les modes "générateur de fréquence de communication", ces procédures doivent être complétées d'un lancement du Timer par mise à 1 de TR2.

TIMER 2 en temporisation

MODE	VALEUR DE T2CON	
	contrôle interne	contrôle externe
Rechargement Auto. 16 bits	00 H (1)	08 H (2)
Capture 16 bits	01 H (1)	09 H (2)
Générateur de fréquence :		
pour émission et réception	34 H	36 H
pour émission seulement	24 H	26 H
pour réception seulement	14 H	16 H

TIMER 2 en compteur

MODE	VALEUR DE T2CON	
	contrôle interne	contrôle externe
Rechargement Auto. 16 bits	02H(1)	0AH(2)
Capture 16 bits	03H(1)	0BH(2)

Notes :
(1) Capture ou rechargement provoqué par dépassement de capacité du TIMER.

SCON : Contrôle du port série

Adresse directe 98H adressable au niveau du bit

9F	9E	9D	9C	9B	9A	99	98
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

mode de fonctionnement du port série :

SM0	SM1	Mode	description	fréquence de communication
0	0	0	registre à décalage	fosc./12
0	1	1	UART 8 bits	variable
1	0	2	UART 9 bits	fosc./64 ou fosc./32
1	1	3	UART 9 bits	variable

SM2 : Communication multiprocesseur possible en mode 2 et 3.
En mode 2 ou 3, si SM2 = 1, alors RI ne sera activé que si le 9^{e} bit est un 1.
En mode 1, si SM2 = 1, alors RI ne sera activé qu'à la réception d'un bit de stop valide.
En mode 0, SM2 doit être positionné à 0.

REN : Mis à 1 par programme REN autorise la réception série.

TB8 : correspond au 9^{e} bit à transmettre en mode 2 ou 3.

RB8 : correspond au 9^{e} bit reçu en mode 2 et 3. En mode 1, si SM2 = 0, RB8 correspond au bit se stop reçu.

TI : est l'indicateur d'émission. Il est automatiquement mis à 1 :
à la transmission du 8^{e} bit en mode 0 ;
à la transmission du bit de stop dans les autres modes.
Avant une opération d'émission, il doit être remis à 0 par programme.

RI : Indicateur de réception. Il est automatiquement mis à 1 :
à la réception de 8^{e} bit en mode 0,
à la réception du bit de stop dans les autres modes
(exception précisée dans la description de SM2).
RI doit être remis à 0 par programme.

Initialisation du port série

MODE	SCON	SM2
0	10H	
1	50H	SM2=0
2	90H	PROCESSEUR UNIQUE
3	D0H	
0	IMPOSSIBLE	SM2=1
1	70H	MULTIPROCESSEUR
2	B0H	
3	F0H	

Définition de la Fréquence de communication FC

Mode 0 :

$$FC = \frac{f_{OSC}}{12}$$

Mode 1 :

FC est déterminée à partir du TIMER 1 configuré en mode "rechargement automatique"

$$TH1 = 256 - \frac{K \times f_{OSC}}{384 \times FC}$$

où

K = si SMOD = 0
K = si SMOD = 1 (SMOD est un bit du registre PCON)

Mode 2 :

si

SMOD = 0 $\quad FC = \frac{f_{OSC}}{64}$

SMOD = 1 $\quad FC = \frac{f_{OSC}}{32}$

Mode 3 :

FC est variable. Même définition que pour le mode 1

Définition de FC à partir du TIMER 2 (8052 seulement) :

Le TIMER 2 est initialisé en mode "recharchement automatique". Il peut être commandé :

1 - par une horloge externe appliquée sur la broche T2

$$FC = \frac{\text{fréquence de dépassement de capacité}}{16}$$

2 - l'horloge externe f_{OSC}

$$FC = \frac{f_{OSC}}{32 \times [65536 - (RCAP2H,RCAP2L)]}$$

d'où

$$RCAP2H,RCAP2L = 65536 - \frac{f_{OSC}}{32 \times FC}$$

Utilisation des interruptions

SOURCE D'INTERRUPTION	INDICATEUR	ADRESSE DU VECTEUR
INT0	IE0	0003H
TIMER 0	TF0	000BH
INT1	IE1	0013H
TIMER 1	TF1	001BH
PORT SERIE	RI+TI	0023H
TIMER 2	TF2+EXF2	002BH

IE : Registre d'autorisation des interruptions

adressage au niveau du bit ADRESSE DIRECTE A8H

AF	AE	AD	AC	AB	AA	A9	A8
EA	-	ET2	ES	ET1	EX1	ET0	EX0

position	symbole	fonction
IE.7	AE	Inhibition de toutes les interruptions.
IE.6	—	sans fonction pour 8051 et 8052
IE.5	ET2	interruption TIMER 2
IE.4	ES	interruption PORT SERIE
IE.3	ET1	interruption TIMER 1
IE.2	EX1	interruption EXTERNE depuis broche INT1
IE.1	ET0	interruption TIMER 0
IE.0	EX0	interruption EXTERNE depuis broche INT0

ordre naturel de prise en compte des interruptions :

ordre	source	
1	IE0	plus haute priorité naturelle
2	TF0	
3	IE1	
4	TF1	
5	RI+TI	
6	TF2+EXF2	plus basse priorité

IP : Registre du niveau de priorité de l'interruption

adressage au niveau du bit adresse directe B8H

position	symbole	fonction
IP.7	—	réservé à une fonction future
IP.6	—	réservé à une fonction future
IP.5	PT2	priorité interruption TIMER 2 (sur 8052)
IP.4	PS	priorité interruption port série
IP.3	PT1	priorité interruption TIMER 1
IP.2	PX1	priorité interruption externe INT1
IP.1	PT0	priorité interruption TIMER 0
IP.0	PX0	priorité interruption externe INT0

Le traitement d'une interruption ne peut être interrompu que par une autre interruption de niveau de priorité supérieur.

PCON : Registre de contrôle de la consommation

Adresse directe 87H

SMOD	—	—	—	GF1	GF0	PD	IDL

SMOD Mis à 1, ce bit assure un doublement de la fréquence de communication du port série obtenue à partir du TIMER 1.

GF1 Indicateur général sans fonction particulière. (1)
GF0 Indicateur général sans fonction particulière. (1)
PD Validation du mode Power-down. (1)
IDL Validation du mode IDLE. (1)

(1) : Ces bits n'existent que dans les versions réalisées en technologie CMOS

JEU D'INSTRUCTIONS DU 8051

Notes sur les symboles utilisés:

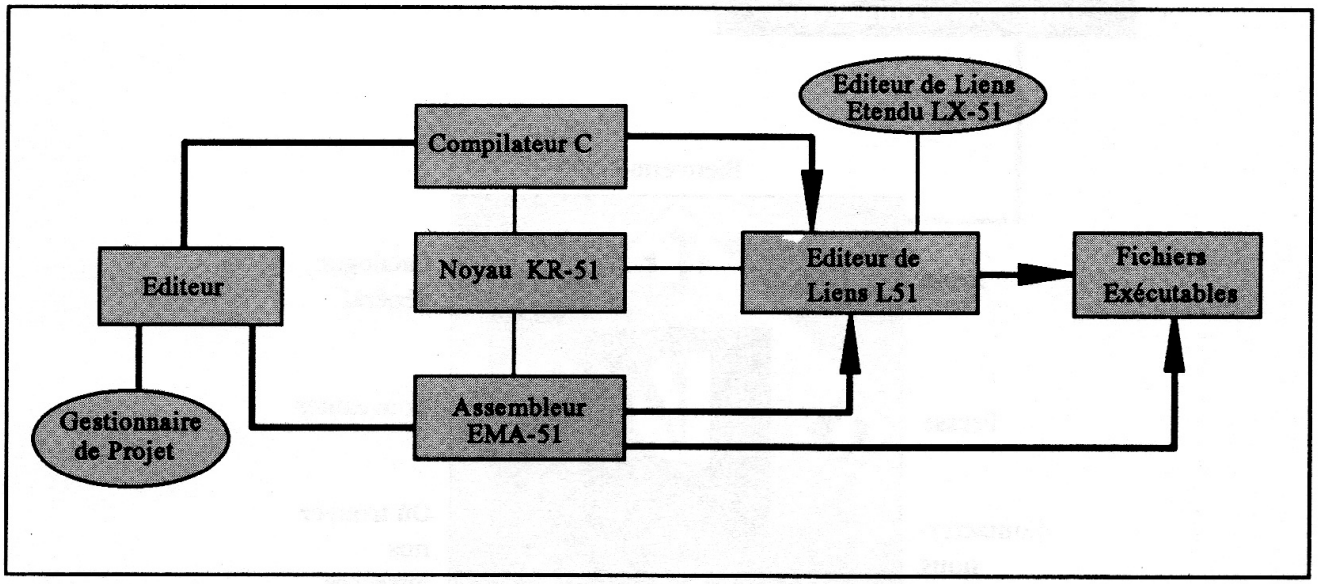
Rn:	Un des registres actifs de R0 à R7
direct :	Adresse d'un des 128 octets de RAM interne, ports, ou SFR.
@Ri :	Adressage indirect par registre R0 ou R1.
#data :	Donnée de 8 bits incluse dans le code exécutable.
# data16 :	Donnée de 16 bits incluse dans le code.
bit :	Adresse au niveau du bit d'un des 128 bits adressables.
rel :	Adresse relative au PC en complément à 2 de +127 à −128.
addr11 :	Une adresse limitée au bloc de 2Ko dans lequel figure l'instruction.
addr16 :	Une adresse sur l'espace de 64 Ko.

Mnémonique		Description	octets	cycle
OPERATIONS DE TRANSFERT				
MOV	A, Rn	Copie, dans A, la valeur du registre	1	1
MOV	A, direct	Copie, dans A, la valeur de l'octet adressé	2	1
MOV	A, @Ri	Copie, dans A, la valeur de l'octet adressé par Ri	1	1
MOV	A, #data	Place dans A la valeur de la donnée immédiate	2	1
MOV	Rn, A	Copie, dans le registre R, la valeur de A	1	1
MOV	Rn, direct	Copie, dans R, la valeur de l'octet adressé	2	2
MOV	Rn, #data	Place dans R, la valeur de la donnée	2	1
MOV	direct, A	Copie, dans l'octet adressé, la valeur de A	2	1
MOV	direct, Rn	Copie, dans l'octet adressé, la valeur de Rn	2	2
MOV	direct, direct	Copie, dans l'octet 1, la valeur de l'octet 2	3	2
MOV	direct, @Ri	Copie, dans l'octet adressé, la valeur adressée par Ri	2	2
MOV	direct, #data	Place dans l'octet adressé, la valeur de la donnée	3	2
MOV	@Ri, A	Copie, dans l'octet adressé par Ri, la valeur de A	1	1
MOV	@Ri, direct	Copie, dans l'octet adressé par Ri, la valeur d'un octet	2	2
MOV	@Ri, #data	Place dans l'octet adressé par Ri, la valeur de la donnée	2	1
MOV	DPTR, #data16	Place dans le registre DPTR, la donnée de 16 bits	3	2
MOVC	A,@A+DPTR	Place dans A, le code d'adresse (DPTR+A)	1	2
MOVC	A,@A+PC	Place dans A, le code d'adresse (PC+A)	1	2
MOVX	A,@Ri	Copie, dans A, la valeur de la RAM externe pointée par Ri	1	2
MOVX	A,@DPTR	Copie, dans A, la valeur de la RAM externe pointée par DPTR	1	2
MOVX	@Ri,A	Copie, dans la RAM externe adressée par Ri, la valeur de A	1	2
MOVX	@DPTR,A	Copie, dans la RAM externe adressée par DPTR, la valeur de A	1	2
PUSH	direct	Sauve dans pile, la valeur de l'octet adressé	2	2
POP	direct	Récupère depuis pile, une valeur placée dans l'octet adressé	2	2
XCH	A,Rn	Echange des contenus de A et de Rn	1	1
XCH	A,direct	Echange des contenus de A et de l'octet adressé	2	1
XCH	A,@Ri	Echange des contenus de A et de la RAM adressée par Ri	1	1
XCHD	A,@Ri	Echange des 4 bits de poids faible de A à RAM pointée par Ri	1	1
OPERATIONS ARITHMETIQUES				
ADD	A,Rn	A= A + Rn	1	1
ADD	A,direct	A= A + valeur de l'octet adressé	2	1
ADD	A,@Ri	A= A + valeur de l'octet adressé par Ri	1	1
ADD	A,#data	A= A + donnée immédiate	2	1
ADDC	A,Rn	A= A + Rn + retenue	1	1
ADDC	A,direct	A= A + valeur de l'octet + retenue	2	1
ADDC	A,@Ri	A= A + valeur de l'octet adressé par Ri + retenue	1	1
ADDC	A,#data	A= A + donnée + retenue	2	1
SUBB	A,Rn	A= A − (Rn + retenue)	1	1
SUBB	A,direct	A= A − (valeur octet + retenue)	2	1
SUBB	A,@Ri	A= A − (valeur octet adressé par Ri + retenue)	1	1
SUBB	A,#data	A= A − (donnée + retenue)	2	1
INC	A	A = A + 1	1	1
INC	Rn	Rn = Rn + 1	1	1
INC	direct	Ajoute 1 à la valeur de l'octet adressé	2	1
INC	@Ri	Ajoute 1 à la valeur de l'octet adressé par Ri	1	1
DEC	A	A = A − 1	1	1
DEC	Rn	Rn = Rn − 1	1	1
DEC	direct	Retranche 1 à la valeur de l'octet adressé	2	1
DEC	@Ri	Retranche 1 à la valeur de l'octet adressé par Ri	1	1
INC	DPTR	Ajoute 1 à la valeur 16 bits de DPTR	1	2
MUL	AB	Multiplication de A par B	1	4
DIV	AB	Division de A par B	1	4
DA	A	Ajustement décimal de A	1	1

JEU D'INSTRUCTIONS DU 8051 (suite)

Mnémonique		Description	octet	cycle
OPERATIONS LOGIQUES				
ANL	A,Rn	A = ET logique entre A et Rn	1	1
ANL	A,direct	A = ET logique entre A et octet	2	1
ANL	A,@Ri	A = ET logique entre A et valeur adressée par Ri	1	1
ANL	A,#data	A = ET logique entre A et donnée	2	1
ANL	direct,A	direct = ET logique entre A et octet adressé	2	1
ANL	direct,#data	direct = ET logique entre donnée et octet adressé	3	2
ORL	A,Rn	A = OU logique entre A et Rn	1	1
ORL	A,direct	A = OU logique entre A et octet	2	1
ORL	A,@Ri	A = OU logique entre A et valeur adressée par Ri	1	1
ORL	A,#data	A = OU logique entre A et donnée	2	1
ORL	direct,A	direct = OU logique entre A et octet adressé	2	1
ORL	direct,#data	direct = OU logique entre donnée et octet adressé	3	2
XRL	A,Rn	A = OU exclusif entre A et Rn	1	1
XRL	A,direct	A = OU exclusif entre A et octet	2	1
XRL	A,@Ri	A = OU exclusif entre A et valeur adressée par Ri	1	1
XRL	A,#data	A = OU exclusif entre A et donnée	2	1
XRL	direct,A	direct = OU exclusif entre A et octet adressé	2	1
XRL	direct,#data	direct = OU exclusif entre donnée et octet adressé	3	2
CLR	A	Mise à 0 de A	1	1
CPL	A	Complémente le contenu de A	1	1
RL	A	Rotation d'un bit, vers gauche, du contenu de A	1	1
RLC	A	Rotation d'un bit, vers gauche, du contenu de (A + retenue)	1	1
RR	A	Rotation d'un bit, vers droite, du contenu de A	1	1
RRC	A	Rotation d'un bit, vers droite, du contenu de (A + retenue)	1	1
SWAP	A	Echange des 4 bits poids fort avec 4 bits poids faible de A	1	1
OPERATIONS BOOLEENNES				
CLR	C	indicateur de retenue mis à 0	1	1
CLR	bit	bit mis à 0	2	1
SETB	C	indicateur de retenue mis à 1	1	1
SETB	bit	bit mis à 1	2	1
CPL	C	complémente l'indicateur de retenue	1	1
CPL	bit	complémente bit	2	1
ANL	C,bit	ET logique entre bit et retenue	2	2
ANL	C,/bit	ET entre complément du bit et C	2	2
ORL	C,bit	OU logique entre bit et C	2	2
ORL	C,/bit	OU entre complément du bit et C	2	2
MOV	C,bit	copie dans retenue C du bit	2	1
MOV	bit,C	copie de la retenue dans bit	2	2
SAUTS ET RUPTURES DE SEQUENCE				
AJMP	addr11	Saut absolu inconditionnel	2	2
LJMP	addr16	Saut long inconditionnel	3	2
SJMP	rel	Saut court par adressage relatif	2	2
JMP	@A+DPTR	Saut indirect à DPTR+A	1	2
JZ	rel	Saut si A = 0	2	2
JNZ	rel	Saut si A différent de 0	2	2
JC	rel	Saut si retenue à 1	2	2
JNC	rel	Saut si retenue à 0	2	2
JB	bit,rel	Saut si bit à 1	3	2
JNB	bit,rel	Saut si bit à 0	3	2
JBC	bit,rel	Saut si bit à 1 et bit=0	3	2
CJNE	A,direct,rel	Saut si A différent de l'octet	3	2
CJNE	A,#data,rel	Saut si A différent de la donnée	3	2
CJNE	Rn,#data,rel	Saut si registre différent de la donnée	3	2
CJNE	@Ri,#data,rel	Saut si octet pointé par Ri ‹› de la donnée	3	2
DJNZ	Rn,rel	décrémente registre et saut si résultat ‹› 0	2	2
DJNZ	direct,rel	décrémente octet et saut si résultat ‹› 0	3	2
GESTION DE SOUS-PROGRAMMES				
ACALL	addr11	Appel absolu de sous-programme	2	2
LCALL	addr16	Appel long de sous-programme	3	2
RET		Retour de sous-programme	1	2
RETI		Retour de sous-programme d'interruption	1	2
AUTRE				
NOP		Pas d'opération	1	1

8051 Une gamme complète d'outils de développement



Outils de génération de code

	Type	Composants supportés	Remarques
SIMICE-51	Simulateur Universel	toute la famille	Simulation de tous les périphériques internes et de l'environnement externe.
EVA-51	Carte d'Evaluation	quasi totalité des composants (en version ROM externe)	Permet la réalisation rapide de prototypes.
TINY-ICE	Mini Emulateur pour le 80C31	80C31	Permet de bénéficier des fonctions d'Emulation les plus usuelles pour un investissement réduit.
MINI-ICE	Mini Emulateur Universel	quasi totalité des composants (en version Rom externe seulement)	Produit milieu de gamme.
PCE-51	Emulateur hautes Performances	quasi totalité des composants (y compris versions EPROM et ROM interne).	Emulation totalement transparente.

Outils de mise au point

RAISONANCE

ZI RUE DES SOURCES 38920 CROLLES FRANCE Tel : 76 08 18 16 Fax : 76 08 09 97

N° éditeur : 044852-(1)-(1,2)-OSB 80-RET
Dépôt légal 1re édit. : 2e trimestre 1993 - Dépôt légal : Novembre 1999
Imprimé en France par I.M.E. - 25110 Baume-les-Dames - N° d'imprimeur : 13801

ÉLECTRONIQUE
Série Technologie électronique

PRINCIPES ÉLECTRONIQUES
TECHNOLOGIE ÉLECTRONIQUE
ÉLECTRONIQUE APPLIQUÉE
DOCUMENTATION ÉLECTRONIQUE

Bernard Odant

MICROCONTRÔLEURS 8051 ET 8052
Description et mise en œuvre

Bien qu'ayant une dizaine d'années, le microcontrôleur 8051 de chez INTEL et les versions qui en dérivent ont obtenu dans le milieu industriel un succès incontestable.

L'unité de la famille est garantie par le fait que toutes les versions sont construites autour d'un même noyau logiciel et matériel. La diversité existe au niveau du choix des périphériques agrégés autour de ce noyau.

L'étude du 8051 et de sa première déclinaison, le 8052, permettront au lecteur de choisir et d'utiliser l'un des nombreux membres de cette famille, dont certaines versions sont encore au stade de la conception.

Cet ouvrage, véritable manuel d'utilisation du 8051 et du 8052, fournit toutes les informations utiles pour découvrir et utiliser ces microcontrôleurs : architecture et ressources internes, jeu d'instructions, modes d'adressage, fonctionnement des programmateurs.

BERNARD ODANT
est professeur en lycée professionnel de la filière Génie électrique.
Il a participé à l'étude et à la réalisation de systèmes de contrôle industriel pour le compte d'un laboratoire de produits cosmétiques.
Les microprocesseurs et microcontrôleurs sont sa spécialité.

ISBN 2 10 004852 X
Code 044852

http://www.dunod.com

DUNOD