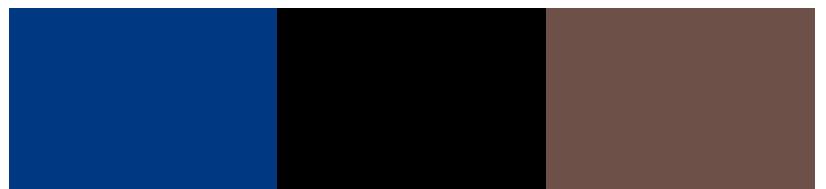


Département Électronique et Physique



**MICROCONTRÔLEUR AT89C51**





# SOMMAIRE

<b>PRÉSENTATION MATÉRIELLE .....</b>	<b>5</b>
<b>1. GÉNÉRALITÉS SUR LES MICROCONTRÔLEURS .....</b>	<b>5</b>
<b>2. PRÉSENTATION DE L'AT89C51 .....</b>	<b>5</b>
<b>2.1. BROCHAGE .....</b>	<b>5</b>
<b>2.2. STRUCTURE INTERNE .....</b>	<b>6</b>
<b>3. MÉMOIRES .....</b>	<b>6</b>
<b>3.1. MÉMOIRE PROGRAMME .....</b>	<b>6</b>
<b>3.2. MÉMOIRE DE DONNÉES .....</b>	<b>7</b>
<b>3.3. MÉMOIRES EXTERNES .....</b>	<b>8</b>
<b>4. REGISTRES DU MICROPROCESSEUR .....</b>	<b>9</b>
<b>4.1. COMPTEUR ORDINAL .....</b>	<b>10</b>
<b>4.2. REGISTRE D'ÉTAT .....</b>	<b>10</b>
<b>4.3. ACCUMULATEUR .....</b>	<b>11</b>
<b>4.4. REGISTRE B .....</b>	<b>11</b>
<b>4.5. POINTEUR DE PILE .....</b>	<b>11</b>
<b>4.6. POINTEUR DE DONNÉE .....</b>	<b>11</b>
<b>5. PORTS D'ENTRÉE/SORTIE .....</b>	<b>12</b>
<b>5.1. PORT 0 .....</b>	<b>12</b>
<b>5.2. PORT 1 .....</b>	<b>13</b>
<b>5.3. PORT 2 .....</b>	<b>13</b>
<b>5.4. PORT 3 .....</b>	<b>14</b>
<b>6. LIAISON SÉRIE .....</b>	<b>14</b>
<b>6.1. MODE 0 .....</b>	<b>15</b>
<b>6.2. MODE 1 .....</b>	<b>15</b>
<b>6.3. MODE 2 .....</b>	<b>16</b>
<b>6.4. MODE 3 .....</b>	<b>16</b>
<b>7. SYSTÈME DE COMPTAGE .....</b>	<b>17</b>
<b>7.1. MODE 0 .....</b>	<b>18</b>
<b>7.2. MODE 1 .....</b>	<b>19</b>
<b>7.3. MODE 2 .....</b>	<b>19</b>
<b>7.4. MODE 3 .....</b>	<b>20</b>
<b>7.5. PRINCIPE D'UTILISATION .....</b>	<b>20</b>

<b>8. INTERRUPTIONS .....</b>	<b>22</b>
<b>8.1. RÉINITIALISATION .....</b>	<b>23</b>
<b>8.2. INTERRUPTIONS EXTERNES .....</b>	<b>24</b>
<b>8.3. INTERRUPTIONS DES COMPTEURS .....</b>	<b>24</b>
<b>8.4. INTERRUPTION DE LA LIAISON SÉRIE .....</b>	<b>24</b>
<b>9. GESTION DE L'ÉNERGIE .....</b>	<b>24</b>
<b>9.1. MODE VEILLE .....</b>	<b>25</b>
<b>9.2. MODE HORS TENSION .....</b>	<b>25</b>
<b>JEU D'INSTRUCTIONS .....</b>	<b>27</b>
<b>1. LANGAGES DE PROGRAMMATION .....</b>	<b>27</b>
<b>2. MODES D'ADRESSAGE .....</b>	<b>27</b>
<b>2.1. ADRESSAGE DE REGISTRE .....</b>	<b>27</b>
<b>2.2. ADRESSAGE IMMÉDIAT .....</b>	<b>28</b>
<b>2.3. ADRESSAGE DIRECT .....</b>	<b>28</b>
<b>2.4. ADRESSAGE INDIRECT .....</b>	<b>29</b>
<b>2.5. ADRESSAGE INDEXÉ .....</b>	<b>30</b>
<b>3. JEU D'INSTRUCTIONS .....</b>	<b>30</b>
<b>4. STRUCTURES EN ASSEMBLEUR .....</b>	<b>34</b>
<b>4.1. PROGRAMME PRINCIPAL .....</b>	<b>34</b>
<b>4.2. STRUCTURES SÉLECTIVES .....</b>	<b>35</b>
<b>4.3. STRUCTURES ITÉRATIVES .....</b>	<b>36</b>
<b>4.4. ROUTINE .....</b>	<b>36</b>
<b>4.5. ROUTINE D'INTERRUPTION .....</b>	<b>37</b>
<b>ANALYSE STRUCTURÉE .....</b>	<b>39</b>
<b>1. INTRODUCTION .....</b>	<b>39</b>
<b>2. ANALYSE DESCENDANTE .....</b>	<b>39</b>
<b>3. ARBRE PROGRAMMATIQUE .....</b>	<b>40</b>
<b>4. TABLEAU DESCRIPTION DES DONNÉES .....</b>	<b>41</b>
<b>5. PSEUDO-CODES .....</b>	<b>41</b>
<b>5.1. STRUCTURE D'UN PROGRAMME .....</b>	<b>42</b>
<b>5.2. VARIABLE .....</b>	<b>42</b>
<b>5.3. AFFECTATION .....</b>	<b>42</b>
<b>5.4. OPÉRATIONS .....</b>	<b>42</b>
<b>5.5. STRUCTURE SÉLECTIVE .....</b>	<b>42</b>
<b>5.6. STRUCTURES ITÉRATIVES .....</b>	<b>42</b>

<b>5.7. APPEL À UN SOUS-PROGRAMME .....</b>	<b>43</b>
<b>5.8. EXEMPLE .....</b>	<b>43</b>
<b>6. LANGAGE ASSEMBLEUR AT89C51 .....</b>	<b>43</b>
<b>RIDE .....</b>	<b>45</b>
<b>1. INSTALLATION .....</b>	<b>45</b>
<b>2. PROJET .....</b>	<b>46</b>
<b>3. FICHIER .....</b>	<b>46</b>
<b>4. PROGRAMME .....</b>	<b>47</b>
<b>4.1. COMMENTAIRE .....</b>	<b>47</b>
<b>4.2. CHAMPS .....</b>	<b>47</b>
<b>4.3. DIRECTIVE D'ASSEMBLAGE .....</b>	<b>48</b>
<b>5. ASSEMBLAGE .....</b>	<b>48</b>
<b>6. DÉBOGUEUR .....</b>	<b>49</b>
<b>6.1. CONFIGURATION .....</b>	<b>49</b>
<b>6.2. MODE DÉBOGAGE .....</b>	<b>50</b>
<b>6.3. BARRE D'OUTILS DU DÉBOGUEUR .....</b>	<b>50</b>
<b>6.4. FENÊTRE <i>Debugger</i> .....</b>	<b>51</b>
<b>6.5. FENÊTRE PROGRAMME .....</b>	<b>52</b>
<b>6.6. FENÊTRE <i>Watches</i> .....</b>	<b>53</b>
<b>6.7. OBSERVATION TEMPORELLE .....</b>	<b>53</b>
<b>6.8. GÉNÉRATEUR DE FONCTION .....</b>	<b>54</b>
<b>ANNEXES .....</b>	<b>55</b>
<b>A.1. REPRÉSENTATION DES NOMBRES .....</b>	<b>55</b>
<b>A.1.1. REPRÉSENTATION BINAIRE .....</b>	<b>55</b>
<b>A.1.2. REPRÉSENTATION HEXADÉCIMALE .....</b>	<b>56</b>
<b>A.1.3. REPRÉSENTATION DES NOMBRES SIGNÉS .....</b>	<b>57</b>
<b>A.1.4. CODAGE DCB .....</b>	<b>57</b>
<b>A.2. MULTIPLE EN BINAIRE .....</b>	<b>58</b>
<b>A.3. FONCTIONS LOGIQUES DE BASE .....</b>	<b>58</b>
<b>A.4. CODE ASCII .....</b>	<b>59</b>
<b>BIBLIOGRAPHIE .....</b>	<b>61</b>



# PRÉSENTATION MATÉRIELLE

## 1. GÉNÉRALITÉS SUR LES MICROCONTROLEURS

Un microcontrôleur est un circuit qui intègre sur la même puce, le processeur, la mémoire programme, la mémoire de données et les dispositifs de communication avec l'environnement extérieur. Comme le montre la figure 1, trois bus (adresse, contrôle et données) permettent aux différentes parties de communiquer.

C'est le composant privilégié des systèmes embarqués (automobile, audio, vidéo, appareils domestiques, etc.) puisqu'il présente les avantages suivants :

- encombrement réduit ;
- faible consommation ;
- circuit imprimé peu complexe ;
- coût réduit en termes de composants, de design et de développement.

Par contre, il est moins puissant et moins rapide que le microprocesseur d'un PC. De plus, il nécessite un matériel pour le programmer, ce qui rend les mises à jour de son programme peu aisées pour l'utilisateur.

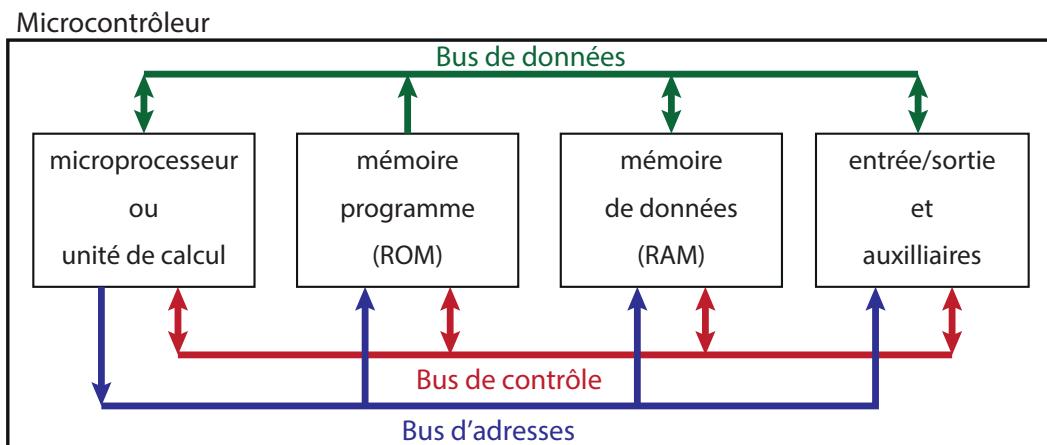


FIGURE 1 – Schéma d'un microcontrôleur

## 2. PRÉSENTATION DE L'AT89C51

### 2.1. BROCHAGE

La famille 8051 de chez Intel est un ensemble de microcontrôleur 8 bits (taille du bus de données) dont l'AT89C51 est une version produite par Atmel. Son brochage pour un boîtier DIL (*Dual In Line*) est donné en figure 2.

L'AT89C51 doit être alimenté sous une tension continue de 5 V entre les broches Vcc et GND. Pour fonctionner, il a aussi besoin d'un quartz (câblé entre les broches XTAL1 et XTAL2) qui permet d'obtenir une fréquence très stable qui cadence les opérations du microcontrôleur. En effet, un cycle machine est constitué de 12 périodes d'horloge, soit :

$$T_{CM} = 12 / f_{OSC}$$

P1.0	1	40	VCC
P1.1	2	39	P0.0 (AD0)
P1.2	3	38	P0.1 (AD1)
P1.3	4	37	P0.2 (AD2)
P1.4	5	36	P0.3 (AD3)
P1.5	6	35	P0.4 (AD4)
P1.6	7	34	P0.5 (AD5)
P1.7	8	33	P0.6 (AD6)
RST	9	32	P0.7 (AD7)
(RXD) P3.0	10	31	EA/VPP
(TXD) P3.1	11	30	ALE/PROG
(INT0) P3.2	12	29	PSEN
(INT1) P3.3	13	28	P2.7 (A15)
(T0) P3.4	14	27	P2.6 (A14)
(T1) P3.5	15	26	P2.5 (A13)
(WR) P3.6	16	25	P2.4 (A12)
(RD) P3.7	17	24	P2.3 (A11)
XTAL2	18	23	P2.2 (A10)
XTAL1	19	22	P2.1 (A9)
GND	20	21	P2.0 (A8)

FIGURE 2 – Brochage DIL de l'AT89C51

## 2.2. STRUCTURE INTERNE

La figure 3 donne la structure interne de l'AT89C51 où :

- la **mémoire programme** est de type EEPROM (*Electrical Erasable Programmable Read Only Memory*) ;
- la **mémoire de données** est de type RAM (*Random Access Memory*) ;
- le **microprocesseur** exécute le programme contenu dans la mémoire programme. L'unité arithmétique et logique (ALU pour *Arithmetic and Logic Unit*) est un calculateur capable de réaliser des opérations logiques (et, ou, etc.) et des calculs binaires (addition, soustraction, multiplication et division) ;
- les **entrées/sorties** et les **auxiliaires** jouent le rôle d'interface entre le microcontrôleur et des événements externes à celui-ci. L'AT89C51 comprend quatre ports d'entrées/sorties (8 bits), une unité d'émission/réception série et deux gestionnaires de temps (*timer*). Les systèmes auxiliaires permettent aussi de gérer le moment de déclenchement des événements externes (interruptions matérielles).

## 3. MÉMOIRES

### 3.1. MÉMOIRE PROGRAMME

La mémoire programme contient le code binaire du programme, qui correspond aux instructions que doit exécuter le microprocesseur. De type EEPROM (*Electrical Erasable Programmable Read Only Memory*), elle ne peut donc être lue que par le microprocesseur. La programmation de cette mémoire se fait grâce à un programmateur.

Elle a une capacité de 4 Kio = 4096 octets (adresses de 0000h à 0FFFh).

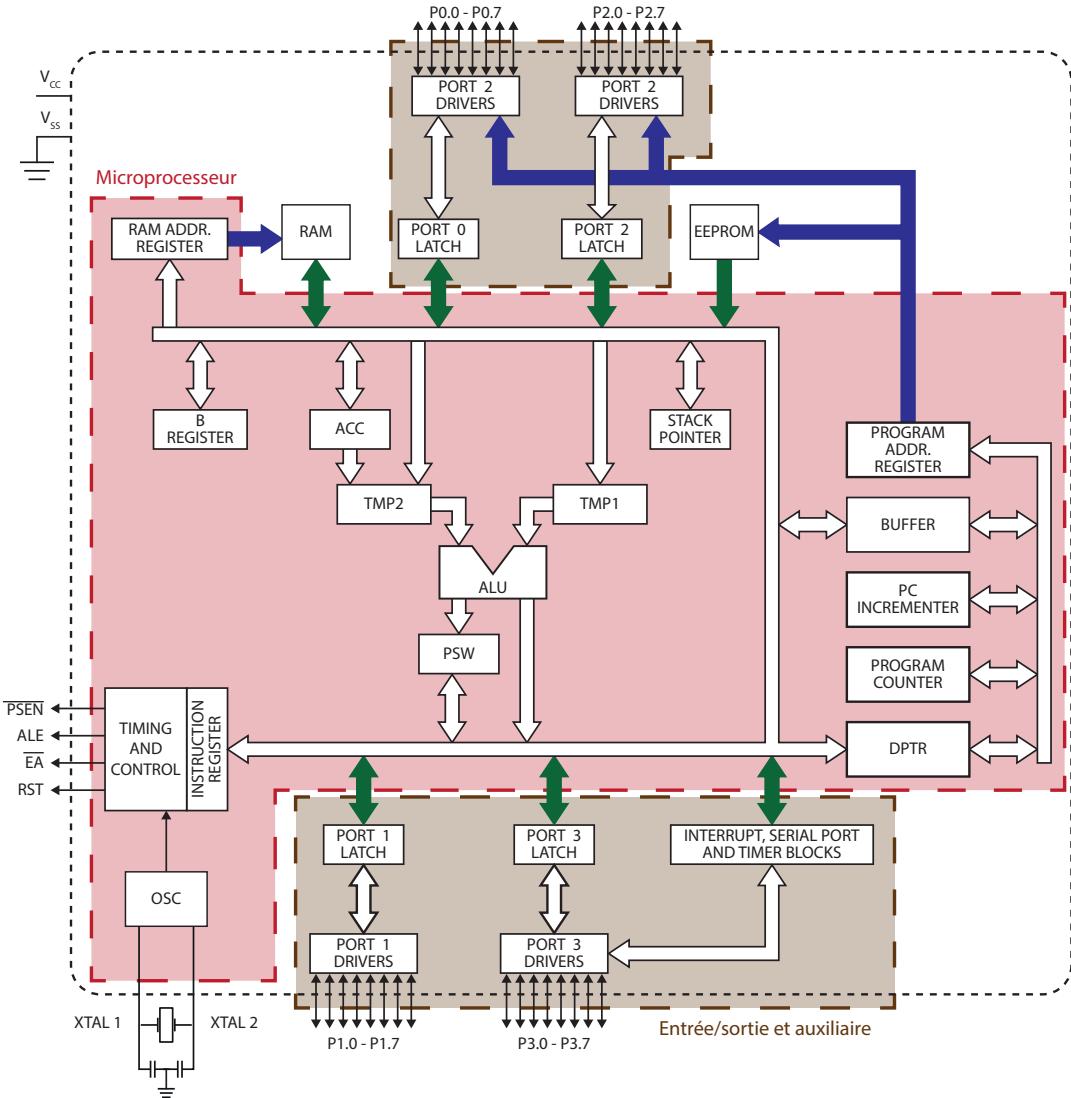


FIGURE 3 – Structure interne de l'AT89C51

### 3.2. MÉMOIRE DE DONNÉES

La mémoire de données sert de stockage pour les résultats de calculs intermédiaires du programme. De type RAM (*Random Access Memory*), elle est donc accessible en lecture et en écriture. Les informations contenues par cette mémoire sont perdues lors d'une coupure de l'alimentation.

D'une capacité de 128 octets, chaque case mémoire est accessible à l'aide d'une adresse (de 00h à 7Fh). En plus, elle est décomposée en 6 parties (voir figure 4) :

- 4 banques (adresses 00h à 1Fh) sélectionnées par les bits RS0 et RS1 du registre d'état. Chaque banque permet d'accéder à 8 registres nommés R0 à R7 (l'adresse effective de ces registres dépend donc des bits RS0 et RS1) ;
- une zone « bits » (adresses mémoire 20h à 2Fh) où chaque bit est directement accessible (128 bits d'adresses 00h à 7Fh) ;
- une zone de 80 octets (adresses 30h à 7Fh).

La distinction entre une adresse octet et une adresse bit est implicite à l'instruction utilisée.

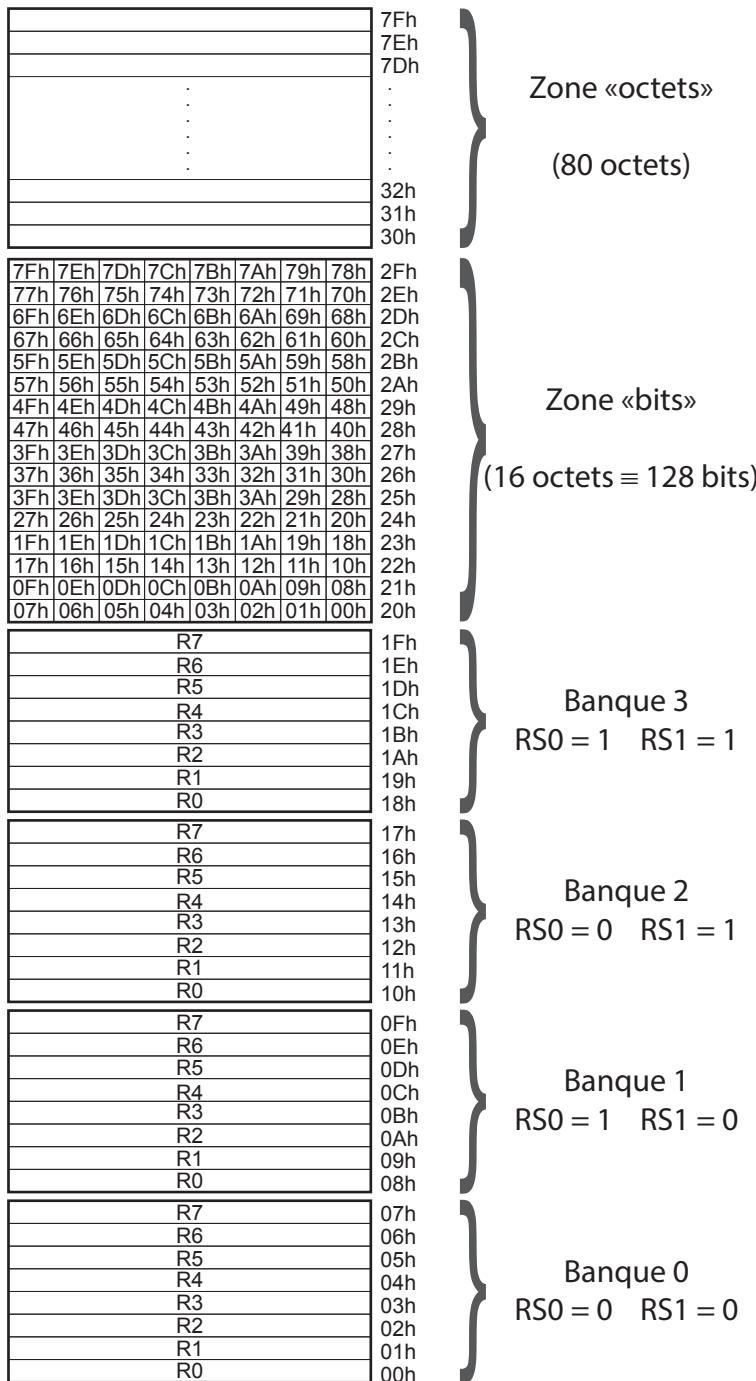


FIGURE 4 – Détail de la mémoire de données de l'AT89C51

### **3.3. MÉMOIRES EXTERNES**

Afin d'étendre les capacités de stockage, il est possible de connecter des mémoires externes qui peuvent être de type programme ou de données.

L'utilisation d'une mémoire programme implique un adressage impératif sur 16 bits tandis que pour une mémoire données, on a le choix entre un adressage sur 8 bits (instruction MOVX @Ri) ou 16 bits (MOVX @DPTR). Dans le cas d'un adressage sur 16 bits, le port 2 est utilisé pour transmettre l'octet de poids fort de l'adresse. Dans tous les cas, l'octet de poids faible est multiplexé dans le temps avec les données sur le port 0. Deux bits du port 3 servent de signaux de commande de lecture (P3.7) ou écriture (P3.6) de donnée.

## 4. REGISTRES DU MICROPROCESSEUR

Comme toutes les adresses à usage spécifique de l'AT89C51, les registres du microcontrôleur (sauf le PC) font partie des registres de fonction spéciale (*Special Function Registers*, SFR). Le détail de ces registres est donné en figure 5 où toutes les adresses sont indiquées sous un format hexadécimal<sup>1</sup> (valeur suivie d'un h). Certains registres sont composés de bits directement accessibles par une adresse propre. Dans ce cas, le bit de poids faible du registre a la même adresse que le registre lui-même et les autres bits ont les adresses suivantes, du bit de poids faible (LSB pour *Low Significant Bit*) au bit de poids fort (MSB pour *Most Significant Bit*). La distinction entre l'adressage du registre et de son bit de poids faible est implicite à l'instruction utilisée.

Registres du microprocesseur																		
A	ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0	E0h									
B	B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0	F0h									
PSW	C	AC	F0	RS1	RS0	OV	-	P	D0h									
DPTR	DPH							83h										
	DPL							82h										
SP																		
PC	PCH							81h										
Système de comptage																		
TIMER 0	TH0							8Ch										
	TL0							8Ah										
TIMER 1	TH1							8Dh										
	TL1							8Bh										
TMOD	GATE	C / $\bar{T}$	M1	M0	GATE	C / $\bar{T}$	M1	M0	89h									
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	88h									
Contrôle des interruptions																		
IE	EA	-	-	ES	ET1	EX1	ET0	EX0	A8h									
IP	-	-	-	PS	PT1	PX1	PT0	PX0	B8h									
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	88h									
Gestion de la liaison série																		
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	98h									
SBUF	Buffer de réception							99h										
SBUF	Buffer d'émission							99h										
Ports d'entrées/sorties																		
P0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	80h									
P1	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	90h									
P2	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	A0h									
P3	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	B0h									
Contrôle d'énergie																		
PCON	SMOD	-	-	-	GF1	GF0	PD	IDL	87h									

FIGURE 5 – Registres de l'AT89C51

1. Voir annexe A.1 Représentation des nombres

#### 4.1. COMPTEUR ORDINAL

Le compteur ordinal ou PC (*Programm Counter*) contient l'adresse de l'instruction suivante de celle en cours d'exécution et permet ainsi de dérouler un programme (enregistré en mémoire programme).

Comme le montre la figure 6, il est composé de 2 octets de 8 bits chacun, nommé PCL et PCH (pour *Low* et *High*). Il ne peut être ni lu, ni écrit directement mais, il est naturellement modifié par les instructions de branchement.



FIGURE 6 – Registre PC

#### 4.2. REGISTRE D'ÉTAT

Le registre d'état ou PSW (*Program Status Word*) permet d'obtenir des informations sur le déroulement du programme.

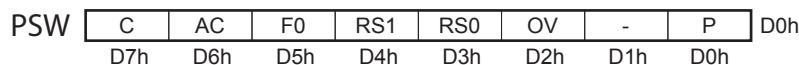


FIGURE 7 – Registre de fonction spéciale PSW

C'est un registre de 8 bits (adresse D0h) où chacun de ces bits est directement accessible (voir figure 7). La signification de chaque bit est :

- C (*Carry*)  $\Rightarrow$  C = 1 indique un débordement sur un octet (retenue) ;
- AC (*Auxillary Carry*)  $\Rightarrow$  AC = 1 indique un débordement sur les 4 bits de poids faible (utilisé par les opérations BCD<sup>1</sup>) ;
- F0 (*Flag 0*)  $\Rightarrow$  F0 est un bit disponible pour le programmeur ;
- RS1 et RS0 (*Register Select*) permettent de sélectionner une banque de la mémoire de données selon le tableau 1 ;

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

TABLEAU 1 – Sélection de la banque à l'aide des bits RS0 et RS1

- OV (*Overflow*)  $\Rightarrow$  OV = 1 indique un débordement de format lors d'une opération sur nombres signés (deux nombres de même signe donnant un résultat de signe opposé) ;
- P (*Parity*)  $\Rightarrow$  P = 0/1 indique un nombre pair/impair de bits à 1 dans l'accumulateur (ACC).

### 4.3. ACCUMULATEUR

L'accumulateur ou ACC (*Accumulator*) est le registre de travail par excellence. Il est utilisé par un grand nombre d'instructions et est alors référencé simplement en tant que A.

C'est un registre de 8 bits (adresse E0h) où chacun de ces bits est directement accessible (voir figure 8).

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

FIGURE 8 – Registre de fonction spéciale ACC

### 4.4. REGISTRE B

Le registre B est utilisé par les opérations multiplication et division. Dans les autres cas, il peut être utilisé en tant que registre classique (espace mémoire quelconque).

C'est un registre de 8 bits (adresse F0h) où chacun de ces bits est directement accessible (voir figure 9).

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

FIGURE 9 – Registre de fonction spéciale B

### 4.5. POINTEUR DE PILE

Une pile (*stack* en anglais) est une structure de données fondée sur le principe « dernier arrivé, premier sorti » (LIFO pour *Last In, First Out*). Elle sert à sauvegarder et donc récupérer des données lorsque plusieurs parties de programme (sous-programmes) utilisent les mêmes ressources (principalement les registres du microcontrôleur).

Le pointeur de pile ou SP (*Stack Pointer*) contient la dernière adresse occupée de la pile. Celle-ci se situe en mémoire de données (la seule accessible en lecture et écriture), le SP est donc un registre de 8 bits (adresse 81h).

Sa gestion est automatique puisque sa valeur est incrémentée avant la sauvegarde d'une nouvelle donnée dans la pile et décrémentée après la restitution de la dernière donnée de la pile.

### 4.6. POINTEUR DE DONNÉE

Le pointeur de donnée ou DPTR (*Data Pointer Register*) est un registre qui permet d'accéder à une donnée en mémoire externe ou en mémoire programme (lecture d'une constante par exemple).

Contenant une adresse, ce registre est composé de 2 octets de 8 bits chacun, nommé DPH (adresse 83h) et DPL (adresse 82h) qui représentent respectivement l'octet de poids fort et faible (voir figure 10).

DPTR	DPH	DPL
	83h	82h

FIGURE 10 – Registre de fonction spéciale DPTR

## 5. PORTS D'ENTRÉE/SORTIE

L'AT89C51 possède 4 ports bidirectionnels de 8 fils accessibles à l'aide des registres P0 à P3 de 8 bits chacun. Leur adresse respective est donnée en figure 11 qui montre que chaque registre est aussi accessible par bit, chaque bit correspondant à un fil du port *ad hoc*.

P0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	80h
	87h	86h	85h	84h	83h	82h	81h	80h	
P1	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	90h
	97h	96h	95h	94h	93h	92h	91h	90h	
P2	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	A0h
	A7h	A6h	A5h	A4h	A3h	A2h	A1h	A0h	
P3	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	B0h
	B7h	B6h	B5h	B4h	B3h	B2h	B1h	B0h	

FIGURE 11 – Registres de fonction spéciale P0 à P3

### 5.1. PORT 0

Le port 0 peut être utilisé comme port d'entrée/sortie standard ou comme bus dédié lors de l'emploi d'une mémoire externe (voir section 3.3). Comme le montre la figure 12, la sortie est à drain ouvert (pas de résistance de tirage). En utilisation standard, le transistor de tirage (transistor supérieur) est bloqué et se comporte comme un interrupteur ouvert.

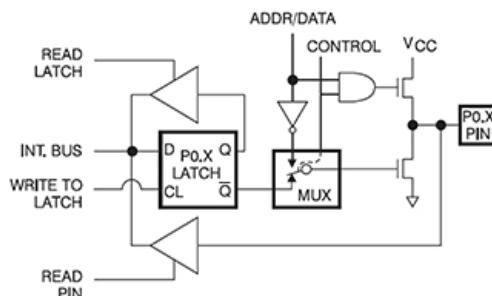


FIGURE 12 – Schéma interne d'une broche du port 0

En sortie, la valeur du bit P0.x (LATCH) est inversée en sortie de bascule D (sortie  $\bar{Q}$ ) qui commande le transistor de contrôle (transistor inférieur). Le tableau 2 donne le détail du fonctionnement en sortie.

Bit P0.x	$\bar{Q}$	Transistor de contrôle	Broche P0.x
0	1	saturé (interrupteur fermé)	masse (0 logique)
1	0	bloqué (interrupteur ouvert)	haute impédance (Hz)

TABLEAU 2 – Fonctionnement d'une broche du port 0 en sortie

Pour fonctionner en entrée, le transistor de contrôle doit être bloqué (dans le cas contraire, la broche est forcée à 0) en positionnant le bit P0.x à 1. Ainsi, la valeur de la broche est envoyée au bus interne (INT. BUS) lors d'une commande de lecture (READ PIN).

## 5.2. PORT 1

Le port 1 est un port d'entrée/sortie standard avec des sorties comportant des résistances de tirage (INTERNAL PULL\_UP), comme le montre la figure 13.

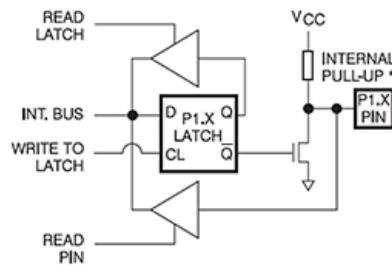


FIGURE 13 – Schéma interne d'une broche du port 1

En sortie, la valeur du bit P1.x (LATCH) est inversée en sortie de bascule D (sortie  $\bar{Q}$ ) qui commande le transistor de contrôle (transistor inférieur). Le tableau 3 donne le détail du fonctionnement en sortie.

Bit P1.x	$\bar{Q}$	Transistor de contrôle	Broche P1.x
0	1	saturé (interrupteur fermé)	masse (0 logique)
1	0	bloqué (interrupteur ouvert)	Vcc (1 logique)

TABLEAU 3 – Fonctionnement d'un broche du port 1 en sortie

Pour fonctionner en entrée, le transistor de contrôle doit être bloqué (dans le cas contraire, la broche est forcée à 0) en positionnant le bit P1.x à 1. Ainsi, la valeur de la broche est envoyée au bus interne (INT. BUS) lors d'une commande de lecture (READ PIN). Lorsque la broche est à 0 (masse), la résistance de tirage évite tout court-circuit de l'alimentation (Vcc) et protège ainsi le circuit.

## 5.3. PORT 2

Le port 2 peut être utilisé comme port d'entrée/sortie standard ou comme bus dédié lors de l'emploi d'une mémoire externe (voir section 3.3).

En utilisation standard, le fonctionnement en entrée et en sortie est identique à celui du port 1, puisqu'il comporte également des résistances de tirage (voir figure 14).

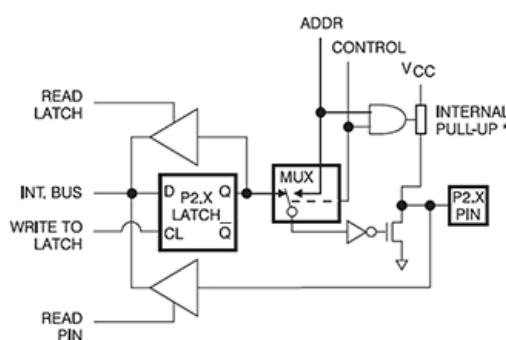


FIGURE 14 – Schéma interne d'une broche du port 2

## 5.4. PORT 3

Le port 3 peut être utilisé comme port d'entrée/sortie standard. Son fonctionnement en entrée et en sortie est alors identique à celui du port 1, puisqu'il comporte également des résistances de tirage (voir figure 15).

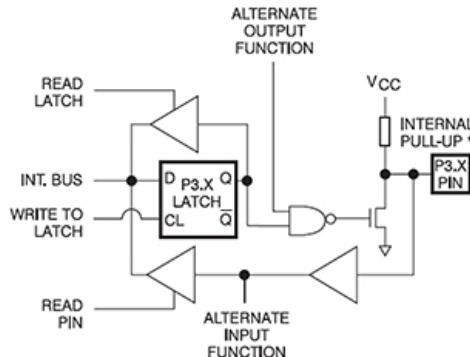


FIGURE 15 – Schéma interne d'une broche du port 3

Les broches du port 3 sont aussi multifonction comme le montre la figure 16, où la signification de chaque bit est :

- $\overline{RD}$  (sortie) ⇒ lecture de la mémoire de données externe ;
- $\overline{WR}$  (sortie) ⇒ écriture en mémoire de données externe ;
- T1 (entrée) ⇒ commande du compteur 1 ;
- T0 (entrée) ⇒ commande du compteur 0 (*timer*) ;
- $\overline{INT1}$  (entrée) ⇒ interruption externe 1 ;
- $\overline{INT0}$  (entrée) ⇒ interruption externe 0 ;
- TXD (sortie) ⇒ transmission de la liaison série ;
- RXD (entrée) ⇒ réception de la liaison série ;

P3	P3.7 $\overline{RD}$ B7h	P3.6 $\overline{WR}$ B6h	P3.5 T1 B5h	P3.4 T0 B4h	P3.3 $\overline{INT1}$ B3h	P3.2 $\overline{INT0}$ B2h	P3.1 TXD B1h	P3.0 RXD B0h	B0h
----	--------------------------------	--------------------------------	-------------------	-------------------	----------------------------------	----------------------------------	--------------------	--------------------	-----

FIGURE 16 – Registre de fonction spéciale P3

## 6. LIAISON SÉRIE

Le principe de la communication série est la transmission de données successives bit après bit, séquentiellement, sur un seul fil. Le port série de l'AT89C51 est de type *full duplex*, ce qui signifie qu'il peut recevoir (broche RXD) et émettre (broche TXD) simultanément. La réception possède un tampon (*buffer*), ce qui autorise la réception d'un second octet avant que l'octet précédent soit lu (cependant la lecture du premier octet doit être réalisée avant la réception complète du second, au risque de perdre le premier).

Les registres d'émission et de réception série sont, tous les deux, accessibles par le registre SBUF (*Serial Buffer* d'adresse 99h) : l'écriture de SBUF charge le registre de transmission série, et la lecture de SBUF permet d'accéder au registre de réception (physiquement séparé).

Le registre SCON (*Serial Control*) permet de contrôler la liaison série. Accessible par bit, il est illustré en figure 17, où la signification de chaque bit est :

- SM0 et SM1 (*Serial Mode*) ⇒ mode de transmission ;
- SM2 ⇒ validation de la communication multiprocesseur (modes 2 et 3) ;
- REN (*Receive Enable*) ⇒ validation de la réception ;
- TB8 *Transmit Bit 8* ⇒ 9<sup>e</sup> bit de donnée transmis (modes 2 et 3) ;
- RB8 *Receive Bit 8* ⇒ 9<sup>e</sup> bit de donnée reçu (modes 2 et 3) ou bit de stop (mode 1) ;
- TI (*Transmit Interrupt Flag*) ⇒ drapeau de fin d'émission (mis à 1 à la fin de l'émission du 8<sup>e</sup> bit dans le mode 0, et au début du bit de stop dans les autres modes) ;
- RI (*Receive Interrupt Flag*) ⇒ drapeau de fin de réception (mis à 1 à la fin de la réception du 8<sup>e</sup> bit dans le mode 0, et à la moitié du bit de stop dans les autres modes).

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

FIGURE 17 – Registre de fonction spéciale SCON

## 6.1. MODE 0

Le mode 0 est sélectionné lorsque SM0 = 0 et SM1 = 0. Il permet d'avoir une transmission half-duplex synchrone, avec :

- la broche RXD (10) sert de ligne de transmission de données en émission et réception ;
- la broche TXD (11) sert à transmettre l'horloge de transmission.

Les données sont transmises dans un format 8 bits (bit de poids faible — LSB — en premier) avec une vitesse de transmission (exprimée en bauds par seconde Bd) fonction de la fréquence d'horloge  $f_{OSC}$  (imposée par le quartz externe) :

$$f_{Mode\ 0} = f_{OSC} / 12$$

## 6.2. MODE 1

Le mode 1 est sélectionné lorsque SM0 = 0 et SM1 = 1. Il permet d'avoir une transmission full-duplex asynchrone avec 10 bits transmis par donnée :

- 1 bit de début (*start*) à 0 ;
- 8 bits de donnée (LSB en premier) ;
- 1 bit de fin (*stop*) à 1.

À la réception, le bit de stop est enregistré dans RB8 de SCON.

La vitesse de transmission est fonction du bit SMOD (bit de poids fort de PCON) et par le taux de dépassement du compteur 1 (*Timer 1*) :

$$f_{Mode\ 1} = \frac{2^{SMOD}}{32} \times (\text{taux de dépassement du Timer 1})$$

La gestion de l'interruption du *Timer 1* doit alors être désactivée.

Lorsque le *Timer 1* est dans le mode compteur avec auto-rechargement (valeur enregistrée dans le registre TH1), la vitesse de transmission est donnée par la formule suivante :

$$f_{\text{Mode 1}} = \frac{2^{\text{SMOD}}}{32} \times \frac{f_{\text{HGL}}}{12 \times [256 - (\text{TH1})]}$$

Il est possible d'obtenir de très faibles vitesses de transmission en configurant le *Timer 1* en compteur 16 bits (mode 1) et en utilisant son interruption avec un rechargement logiciel sur 16 bits. Le tableau 4 liste les vitesses de transmission communément utilisées et la manière de les obtenir à l'aide du *Timer 1*.

Vitesse de transmission	$f_{\text{osc}}$	SMOD	Timer1		
			C/T	Mode	Chargement
Mode 0 : 1 MBd max	12 MHz	x	x	x	x
Mode 2 : 375 kBd max	12 MHz	1	x	x	x
Mode 1 & 3 : 62,5 kBd max	12 MHz	1	0	2	FFh
19,2 kBd	11,059 MHz	1	0	2	FDh
9,6 kBd	11,059 MHz	0	0	2	FDh
4,8 kBd	11,059 MHz	0	0	2	FAh
2,4 kBd	11,059 MHz	0	0	2	F4h
1,2 kBd	11,059 MHz	0	0	2	E8h
137,5 Bd	11,059 MHz	0	0	2	1Dh
110 Bd	6 MHz	0	0	2	72h
110 Bd	12 MHz	0	0	1	FEEBh

TABLEAU 4 – Vitesses de transmission usuelles générées par le *Timer 1*

### 6.3. MODE 2

Le mode 2 est sélectionné lorsque SM0 = 1 et SM1 = 0. Il permet d'avoir une transmission full-duplex asynchrone avec 11 bits transmis par donnée :

- 1 bit de start à 0 ;
- 8 bits de donnée (LSB en premier) ;
- 1 bit programmable  $\Rightarrow$  bit TB8 de SCON (typiquement utilisé en bit de parité ou en deuxième bit de stop) ;
- 1 bit de stop à 1.

À la réception, le bit programmable est enregistré dans RB8 de SCON.

La vitesse de transmission est fonction du bit SMOD (bit de poids fort de PCON) et de la fréquence d'horloge  $f_{\text{OSC}}$  :

$$f_{\text{Mode 2}} = \frac{2^{\text{SMOD}}}{64} \times f_{\text{OSC}}$$

### 6.4. MODE 3

Le mode 3 est sélectionné lorsque SM0 = 1 et SM1 = 1. Le fonctionnement est identique au mode 2 : transmission full-duplex asynchrone sur 11 bits par donnée, avec une vitesse de transmission identique au mode 1.

## 7. SYSTÈME DE COMPTAGE

L'AT89C51 possède deux systèmes de comptage, nommés *timer* 0 et *timer* 1 au fonctionnement similaire. Chaque *timer* utilise un registre de 16 bits (TIMER 0 et TIMER 1) divisé en 2 registres de 8 bits TLx et THx (le x doit être remplacé par 0 ou 1 en fonction du *timer* utilisé), comme illustré en figure 18.

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

FIGURE 18 – Registres de fonction spéciale TIMER 0 et TIMER 1

Ces systèmes peuvent être utilisés en :

- minuteur (*timer*)  $\Rightarrow$  le registre TIMER x est incrémenté à chaque cycle machine. Le taux de comptage est alors de :

$$f_{\text{timer}} = f_{\text{OSC}} / 12$$

- compteur d'événement (*counter*)  $\Rightarrow$  le registre TIMER x est incrémenté à chaque front descendant de l'entrée Tx (P3.4 et P3.5). Le taux de comptage maximum est de :

$$f_{\text{counter max}} = f_{\text{OSC}} / 24$$

Le registre TCON (*Timer/Counter Control*) contient des bits de contrôle des compteurs. Accessible par bit, il est illustré en figure 19, où :

- TFx (*Timer Flag*)  $\Rightarrow$  drapeau de dépassement de TIMER x
  - TFx passe à 1 lors du passage à 0 du registre TIMER x ;
  - TFx est mis à 0 lors de la prise en compte de l'interruption du *timer* x ;
- TRx (*Timer Run Control*)  $\Rightarrow$  marche/arrêt du *timer* x
  - TRx = 0  $\Rightarrow$  arrêt du *timer* x ;
  - TRx = 1  $\Rightarrow$  démarrage du *timer* x ;
- IEx (*Interrupt Edge Flag*)  $\Rightarrow$  drapeau d'interruption
  - IEx est mis à 1 lorsqu'une demande d'interruption sur la broche  $\overline{\text{INTx}}$  (P3.2 et P3.3) est détectée ;
  - IEx est mis à 0 lors de la prise en compte de l'interruption du *timer* x ;
- ITx (*Interrupt Type Control*)  $\Rightarrow$  contrôle le mode de fonctionnement de la broche  $\overline{\text{INTx}}$ 
  - ITx = 0  $\Rightarrow$   $\overline{\text{INTx}}$  actif sur un niveau bas (0 logique) ;
  - ITx = 1  $\Rightarrow$   $\overline{\text{INTx}}$  actif sur front descendant.

TCON	TF1 8Fh	TR1 8Eh	TF0 8Dh	TR0 8Ch	IE1 8Bh	IT1 8Ah	IE0 89h	IT0 88h	88h
------	------------	------------	------------	------------	------------	------------	------------	------------	-----

FIGURE 19 – Registre de fonction spéciale TCON

Le registre TMOD (*Timer/Counter Mode Control*) permet de contrôler le mode de fonctionnement des compteurs. Il n'est pas accessible par bit, et, comme illustré en figure 20, les 4 bits de poids faible sont associés au *timer* 0 et les 4 bits de poids fort au *timer* 1, avec :

- GATE  $\Rightarrow$  contrôle externe
  - GATE = 0  $\Rightarrow$  le *timer* est valide (il compte) lorsque le bit TR<sub>x</sub> = 1 ;
  - GATE = 1  $\Rightarrow$  le *timer* est valide lorsque le bit TR<sub>x</sub> = 1 et lorsque la broche INT<sub>x</sub> est à un niveau haut (1 logique) ;
- C/T (Timer or Counter Selector)  $\Rightarrow$  sélection du type de fonctionnement
  - C/T = 0  $\Rightarrow$  fonctionnement en minuteur (*timer*) ;
  - C/T = 1  $\Rightarrow$  fonctionnement en compteur d'évènement (*counter*) ;
- M1 et M0 (*Mode*) : choix du mode de fonctionnement.

TMOD	GATE	C/T	M1	M 0	GATE	C/T	M1	M 0	89h
<i>timer 1</i>					<i>timer 0</i>				

FIGURE 20 – Registre de fonction spéciale TMOD

## 7.1. MODE 0

Le mode 0 est sélectionné lorsque M1 = 0 et M0 = 0. Le *timer* fonctionne alors en compteur 8 bits avec un prédiviseur (*prescaler*) de 1/32. La figure 21 illustre alors le fonctionnement du *timer* 1, le *timer* 0 fonctionnant sur le même principe (avec les broches et bit associés à ce compteur).

Dans ce mode, le registre TIMER x est configuré comme un registre 13 bits avec les 8 bits de TH<sub>x</sub> (octet de poids fort) et les 5 bits de poids faible de TL<sub>x</sub> ; les 3 bits de poids forts de TL<sub>x</sub> ont une valeur indéterminée et doivent être ignorés. Le registre TL<sub>x</sub> joue alors le rôle du prédiviseur par 32 (2<sup>5</sup>) et TH<sub>x</sub> celui du compteur 8 bits.

La mise à 1 du bit TR<sub>x</sub> démarre le compteur mais ne l'initialise pas à zéro. Le passage de FFh à 00h de TH<sub>x</sub> (seul registre utilisé dans ce mode) valide le drapeau de dépassement du *timer* x (TF<sub>x</sub> = 1).

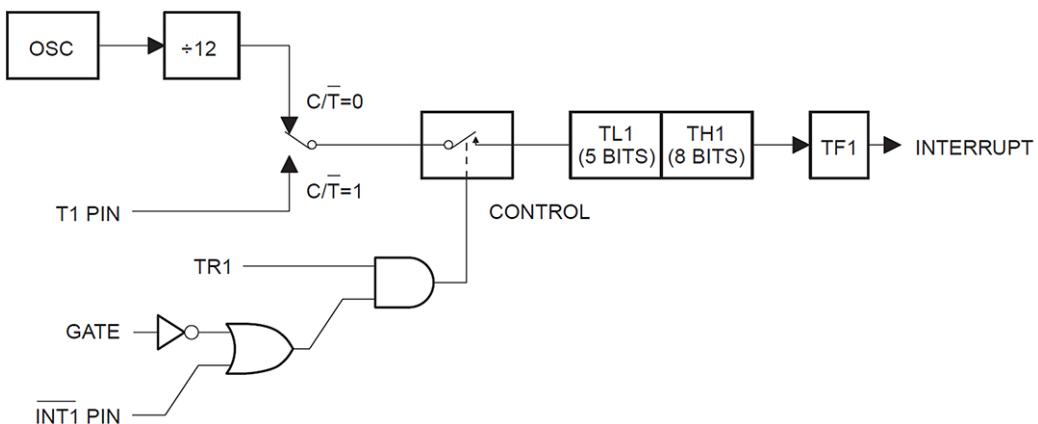


FIGURE 21 – Schéma de fonctionnement du *timer* 1 en mode 0

## 7.2. MODE 1

Le mode 1 est sélectionné lorsque  $M1 = 0$  et  $M0 = 1$ . Le *timer* fonctionne alors en compteur 16 bits, comme illustré en figure 22 (les *timer* 0 et 1 fonctionnent sur le même principe, avec les broches et bits associés).

La mise à 1 du bit  $TRx$  démarre le compteur mais ne l'initialise pas à zéro. Le passage de  $FFFFh$  à  $0000h$  du registre  $TIMER x$  valide le drapeau de dépassement du *timer*  $x$  ( $TFx = 1$ ).

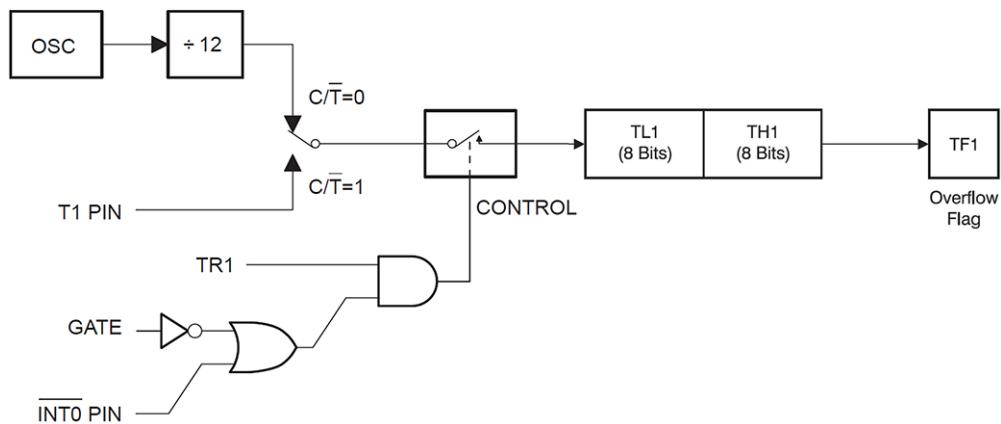


FIGURE 22 – Schéma de fonctionnement du *timer* 1 en mode 1

## 7.3. MODE 2

Le mode 2 est sélectionné lorsque  $M1 = 1$  et  $M0 = 0$ . Le *timer* fonctionne alors en compteur 8 bits avec recharge automatique, comme illustré en figure 23 (les *timer* 0 et 1 fonctionnent sur le même principe, avec les broches et bit associés).

Le dépassement du registre  $TLx$  (passage de  $FFh$  à  $00h$ ) fait passer le bit  $TFx$  à 1 et recharge  $TLx$  avec le contenu de  $THx$ , la valeur de ce dernier restant inchangée.

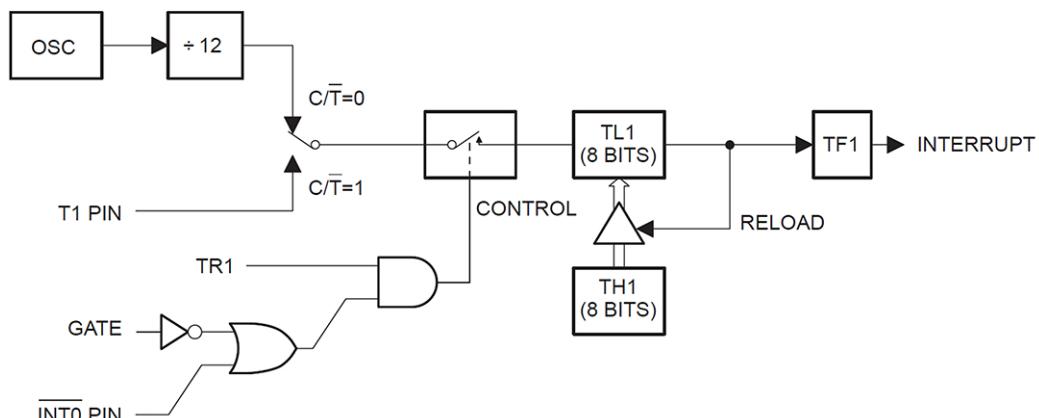


FIGURE 23 – Schéma de fonctionnement du *timer* 1 en mode 2

## 7.4. MODE 3

Le mode 3 est sélectionné lorsque M1 = 1 et M0 = 1. Dans ce mode, Le *timer* 0 se comporte comme deux compteurs 8 bits (TH0 et TL0) et le *timer* 1 est arrêté (équivalent à TR1 = 0).

Comme le montre la figure 24, TL0 utilise les bits de contrôle du *timer* 0 : GATE, C/T, TF0, TR0 et la broche  $\overline{\text{INT}0}$ . Le registre TH0 est bloqué dans une fonction minuteur (comptage avec un taux de  $f_{\text{OSC}}/12$ ) et utilise les bits TR1 et TF1. TH0 contrôle donc l'interruption du *timer* 1.

Quand le *timer* 0 est dans le mode 3, le *timer* 1 peut être démarré et arrêté en basculant or et dans le mode 3. Ainsi, il peut toujours être utilisé pour la liaison série ou tout autre application ne nécessitant pas une interruption.

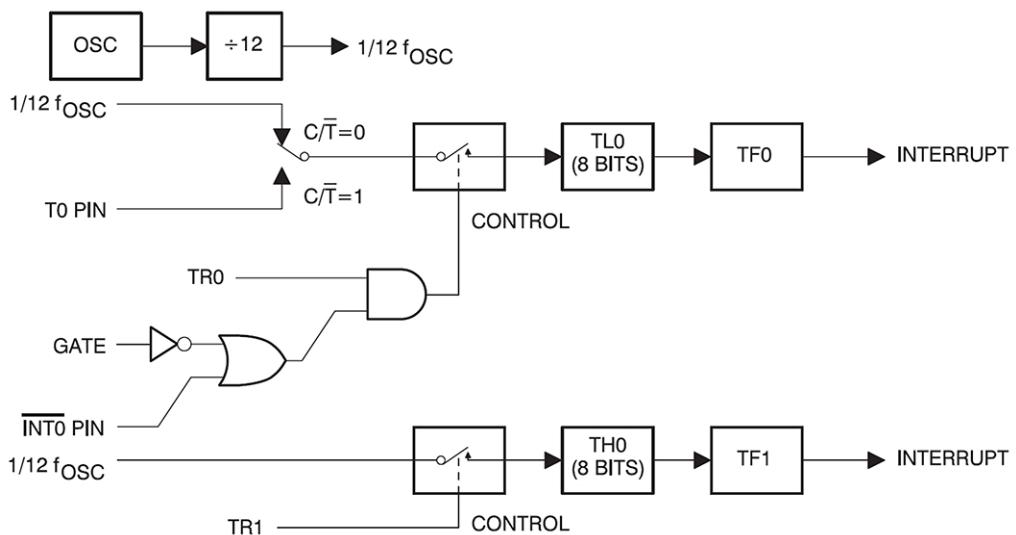


FIGURE 24 – Schéma de fonctionnement du *timer* 1 en mode 3

## 7.5. PRINCIPE D'UTILISATION

L'utilisation d'un compteur nécessite plusieurs opérations :

1. Définir le mode de fonctionnement (TMOD).
2. Initialiser la valeur du compteur (THx et TLx).
3. Initialiser le drapeau (TFx).
4. Démarrer le compteur (TRx).
5. Scruter le dépassement (TFx).

En cas de comptage en boucle, il faudra en plus :

6. Arrêter le compteur (TRx).
7. Refaire à partir de l'opération 2.

Lorsqu'on souhaite compter un certains nombre d'évènements, il faut charger le compteur à une valeur définie (étape 2). Puisque l'activation du drapeau TFx arrive lorsque la valeur du registre est nulle, la valeur du chargement est le complément de la valeur désirée (voir figure 25).

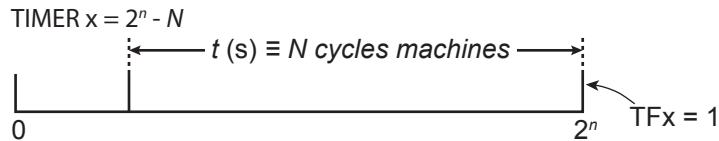


FIGURE 25 – Principe du chargement d'un compteur

Soit l'exemple suivant : compter  $t = 100 \mu\text{s}$  avec un compteur de  $n = 8$  bits et un quartz de  $f_{\text{OSC}} = 6 \text{ MHz}$ .

- un compteur 8 bits implique un nombre total de valeurs :

$$2^n = 2^8 = 256$$

- la durée d'un cycle machine correspond à 12 périodes d'horloge, soit avec un quartz de 6 MHz :

$$T_{\text{CM}} = 12 / f_{\text{OSC}} = 2 \mu\text{s}$$

- le nombre de cycles machines correspondant au temps désiré est :

$$N = t / T_{\text{CM}} = 50$$

- la valeur de chargement est alors :

$$\text{TIMER } x = 2^n - N = 206 = \text{CEh}$$

Si l'on souhaite répéter consécutivement ce comptage alors, pour obtenir un comptage précis, il faut tenir compte de deux paramètres supplémentaires :

1. Le temps de dépassement qui correspond au temps entre l'activation du drapeau ( $\text{TFx} = 1$ ) et l'arrêt effectif du compteur. Ce temps correspond à la valeur du compteur à son arrêt :  $N_D$ .
2. Le temps de recharge qui correspond au nombre de cycles machines entre l'arrêt ( $\text{TRx} = 0$ ) et le redémarrage ( $\text{TRx} = 1$ ) du compteur. Ce nombre  $N_R$  dépend des instructions placées entre ces deux opérations et doit donc être calculé au cas par cas.

Ceci est illustré en figure 26, la valeur de chargement effective doit donc être :

$$\text{TIMER } x = 2^n - (N - N_D - N_R) = 2^n - N + N_D + N_R$$

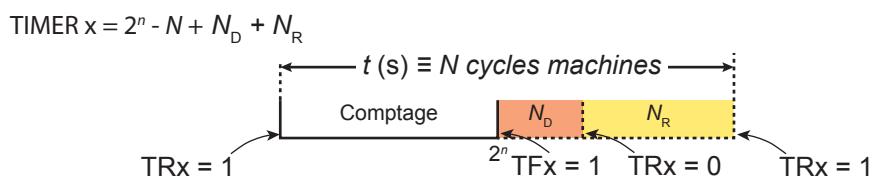


FIGURE 26 – Principe du chargement d'un compteur en boucle et précis

## 8. INTERRUPTIONS

Une interruption est un arrêt temporaire de l'exécution normale d'un programme par le microprocesseur, suite à un évènement interne ou externe, afin d'exécuter un autre programme, appelé routine d'interruption.

L'AT89C51 possède six sources d'interruption : deux externes (broches  $\overline{\text{INT}0}$  et  $\overline{\text{INT}1}$ ), un pour la liaison série, un pour chaque compteur (*timer 0* et *timer 1*) et un pour la réinitialisation (broche RST pour *Reset*). Chaque interruption (sauf le *reset*) est associée à un drapeau, dont le détail est donné en tableau 5. Les interruptions (IT) sont dites vectorisées, c'est-à-dire que chaque IT est associée à une adresse fixe (voir tableau 5). Lors de la prise en compte d'une IT, le compteur ordinal (PC) sera chargé avec cette adresse, où devra se situer la première instruction de la routine d'interruption (en mémoire programme).

Interruption	Source	@ de saut
<i>Reset</i> système	broche RST	0000h
$\overline{\text{INT}0}$	bit IE0 de TCON	0003h
<i>Timer 0</i>	bit TF0 de TCON	000Bh
$\overline{\text{INT}1}$	bit IE1 de TCON	0013h
<i>Timer 1</i>	bit TF1 de TCON	001Bh
Port série	bits TI ou RI de SCON	0023h

TABLEAU 5 – Table des vecteurs d'interruption

Chaque interruption peut être masquée (ne pas être prise en compte) ou validée en configurant le registre IE. Accessible par bit, il est illustré en figure 27, où la mise à 0 d'un bit permet de masquer l'IT correspondant et la mise à 1 de la valider :

- EA ⇒ masquage/validation générale (toutes les IT) ;
- ES ⇒ masquage/validation de l'IT de la liaison série ;
- ETx ⇒ masquage/validation de l'IT du *timer x* ;
- EXx1 ⇒ masquage/validation de l'IT externe x ( $\overline{\text{INT}x}$ ).

IE	[	EA	-	-	ES	ET1	EX1	ET0	EX0	]	A8h
		AFh	AEh	ADh	ACh	ABh	AAh	A9h	A8h		

FIGURE 27 – Registre de fonction spéciale IE

L'AT89C51 peut gérer deux niveaux de priorité (0 et 1). Le niveau de priorité de chaque IT est défini par la valeur du bit correspondant du registre IP, accessible par bit (voir figure 28) :

- PS ⇒ priorité de l'IT de la liaison série ;
- PTx ⇒ priorité de l'IT du *timer x* ;
- PXx ⇒ priorité de l'IT externe x ( $\overline{\text{INT}x}$ ).

IP	[	-	-	-	PS	PT1	PX1	PT0	PX0	]	B8h
		BFh	BEh	BDh	BCh	BBh	BAh	B9h	B8h		

FIGURE 28 – Registre de fonction spéciale IP

Une routine d'IT ne peut être interrompue que par une demande d'IT de priorité strictement supérieure : une IT de niveau 0 peut être interrompue par une IT de niveau 1 et une IT de niveau 1 ne peut donc pas être interrompue. Si une IT n'est pas prise en compte « immédiatement », elle est toujours validée (drapeau positionné à un) et sera donc gérée à la fin de la routine d'IT en cours. Si plusieurs IT de même priorité doivent être gérées en même temps, il existe une seconde structure de priorité déterminée par la séquence d'interrogation, comme suit :

1. IE0 (plus prioritaire)
2. TF0
3. IE1
4. TF1
5. TI ou RI (moins prioritaire)

La seule exception est le *reset* qui est prioritaire sur toutes les autres exceptions et qui ne peut pas être masqué.

Lorsqu'une IT est acceptée, les opérations suivantes se déroulent de manière chronologique :

1. Exécution de l'instruction en cours.
2. Sauvegarde du PC est sauvegardé dans la pile (pré-incrémentation du SP).
3. Sauvegarde, en interne, de l'état d'IT courant (priorité).
4. Blocage des IT de même priorité que l'IT acceptée.
5. Chargement du PC avec l'adresse de l'IT (vectorisation).
6. Exécution de la routine d'IT.

La fin de la routine d'IT est marquée par l'instruction RETI. Le retour au programme interrompu se fait par les opérations suivantes :

1. Restauration du PC depuis la pile (post-décrémentation du SP).
2. Restauration de l'ancien niveau de priorité.

## 8.1. RÉINITIALISATION

La réinitialisation de l'AT89C51 ne peut se faire que par la broche RST (*reset*), il n'y a pas de *reset* logiciel. L'IT apparaît lorsque cette broche est à un niveau haut (+5 V) pendant au moins 2 cycles machines (lorsque l'oscillateur du microcontrôleur fonctionne).

La réinitialisation consiste en :

- PC = 0000h ;
- tous les SFR<sup>2</sup> = 00h (sauf P0 à P3) ;
- P0 à P3 = FFh ⇒ les ports sont configurés en entrée ;
- SP = 07h ⇒ la première adresse utilisée pour la pile est 08h (incrémentation de SP avant sauvegarde dans la pile) et correspond au registre R0 de la banque 1 ;
- SBUF est indéterminé.

La RAM n'est pas affectée par un *reset* (indéterminée à la mise sous tension).

---

2. Les bits non utilisés des SFR ont une valeur indéterminée au *reset*.

## 8.2. INTERRUPTIONS EXTERNES

Une interruption externe est provoquée par un évènement sur la broche  $\overline{\text{INT0}}$  ou  $\overline{\text{INT1}}$  (multiplexées avec P3.2 et P3.3). Cet évènement est configuré avec le bit ITx de TCON (cf. figure 19, page 17) :

- $\text{ITx} = 0 \Rightarrow \overline{\text{INTx}}$  actif sur un niveau bas ;
- $\text{ITx} = 1 \Rightarrow \overline{\text{INTx}}$  actif sur un front descendant.

À l'arrivée de l'évènement, le bit IEx de TCON passe à 1. Lors de la vectorisation :

- $\text{ITx} = 0 \Rightarrow \text{IEx}$  recopie la valeur de INTx (complément de l'entrée  $\overline{\text{INTx}}$ ) ;
- $\text{ITx} = 1 \Rightarrow \text{IEx} = 0$ .

## 8.3. INTERRUPTIONS DES COMPTEURS

Les drapeaux associés à ces IT sont les bits TFx de TCON (cf. figure 19, page 17) dont le fonctionnement dépend du mode de fonctionnement des compteurs :

- mode 0 à 2 de *timer x*
  - dépassement de TIMER x  $\Rightarrow \text{TFx} = 1$
- mode 3 de *timer 0*
  - dépassement de TL0  $\Rightarrow \text{TF0} = 1$
  - dépassement de TH0  $\Rightarrow \text{TF1} = 1$

Lors de la vectorisation, le drapeau du *timer ad hoc* est mis à 0 ( $\text{TFx} = 0$ ).

## 8.4. INTERRUPTION DE LA LIAISON SÉRIE

Cette interruption est provoquée lorsque le bit TI ou RI de SCON passe à 1 (cf. figure 17, page 15). Comme il n'est pas possible de distinguer une IT provenant de la réception ou de l'émission d'une donnée, la vectorisation n'affecte donc aucun drapeau (TI et RI). La routine d'IT doit donc faire la distinction entre ces deux cas et mettre à zéro le drapeau adéquat.

# 9. GESTION DE L'ÉNERGIE

L'AT89C51 possède deux modes d'économie d'énergie : veille (*idle*) et hors tension *power down*. Ces modes sont activés par la mise à 1 d'un bit du registre PCON. La figure 29 illustre ce registre qui n'est pas accessible par bit et dont la signification de chaque bit est :

- SMOD (*Serial Mode*)  $\Rightarrow$  doublement de la vitesse de transmission de la liaison série ;
- GFx (*General Flag*)  $\Rightarrow$  drapeau d'usage général ;
- PD *Power Down*  $\Rightarrow$  mode hors tension ;
- IDL (*Idle*)  $\Rightarrow$  mode veille.

PCON	SMOD	-	-	-	GF1	GF0	PD	IDL	87h
------	------	---	---	---	-----	-----	----	-----	-----

FIGURE 29 – Registre de fonction spéciale PCON

## 9.1. MODE VEILLE

Une instruction qui définit le bit IDL à 1 est la dernière instruction exécutée avant le début du mode veille. Dans ce mode, le signal d'horloge interne est désactivé pour le processeur, mais pas pour les interruptions, les compteurs et la liaison série. L'état du processeur est préservé dans son intégralité : PC, SP, PSW, ACC et tous les autres registres conservent leurs données au repos. Les broches des ports contiennent les états logiques qu'elles avaient au moment de l'activation du mode veille.

Il y a deux manières de terminer l'inactivité. L'activation de toute interruption activée entraîne la mise à 0 du bit IDL, mettant ainsi fin au mode veille. L'interruption est alors traitée, et après l'instruction RETI de la routine d'IT, la prochaine instruction exécutée est celle qui suit l'instruction qui place le périphérique en veille (valeur du PC au moment de la mise en veille). Les bits GF0 et GF1 peuvent être utilisés pour indiquer si une interruption s'est produite pendant le fonctionnement normal ou pendant le mode veille. L'autre façon de terminer le mode veille est une réinitialisation matérielle (*reset*), le signal sur la broche RST (maintenu pendant deux cycles machines) efface le bit IDL directement. À ce moment, le processeur reprend l'exécution du programme là où il s'est arrêté, c'est-à-dire à l'instruction qui suit celle qui a appelé le mode veille.

## 9.2. MODE HORS TENSION

Une instruction qui définit le bit PD à 1 est la dernière instruction exécutée avant le début du mode de mise hors tension. Dans ce mode, l'oscillateur est arrêté et donc avec lui, toutes les fonctions du microcontrôleur. Cependant, les données de la RAM et les registres de fonctions spéciales sont maintenus. Les broches des ports génèrent les valeurs détenues par leurs SFR respectifs.

La sortie du mode hors tension ne peut se faire qu'avec une réinitialisation matérielle (*reset*) ; ce qui redéfinit tous les SFR, mais ne modifie pas la RAM.

En mode hors tension, la tension d'alimentation peut être réduite à 2 V. Cependant, cette tension ne doit pas être réduite avant que le mode hors tension ne soit appelé et elle doit être rétablie à son niveau de fonctionnement normal avant la fin du mode hors tension.



# JEU D'INSTRUCTIONS

## 1. LANGAGES DE PROGRAMMATION

Écrit en binaire, le **langage machine** est le seul compréhensible par un microprocesseur. Une instruction machine est constituée d'un code opératoire (CO) et éventuellement d'une partie auxiliaire (AUX). Le CO contient l'opération à effectuer et parfois des informations sur le ou les registres manipulés. L'AUX complète le CO afin de réaliser l'opération souhaitée : donnée, adresse, etc. Dans le cas de l'AT89C51, le code opératoire a une taille de 1 octet (soit 256 CO différents) et la partie auxilliaire de 0 à 2 octets.

Souvent représenté en hexadécimal, le langage machine est peu aisément programmable. Le **langage assembleur** est alors utilisé car il y a une correspondance directe entre ces deux langages. En effet, une instruction assembleur est traduite en une instruction machine à l'aide d'un logiciel qui se nomme assembleur (l'opération de traduction se nomme l'assemblage). La programmation est plus aisée puisqu'il est composé d'un mnémonique (groupe de lettres permettant d'identifier l'opération effectuée) et éventuellement d'opérandes, maximum deux dans le cas de l'AT89C51. Lorsque l'instruction assembleur nécessite deux opérandes, elles sont nommées destination (DST) et source (SRC), placées dans cet ordre et séparées par une virgule :

MNEMONIQUE DST , SRC

Le langage machine et le langage assembleur sont appelés langages de bas niveau, puisqu'ils permettent de manipuler explicitement les registres, les adresses mémoires, etc. Les **langages de haut niveau** (Python, C, etc.) ont l'avantage d'une programmation plus rapide et sans connaissance particulière du microprocesseur au prix d'aucune maîtrise en terme de place mémoire ou de durée d'exécution. En effet, une instruction haut niveau correspond à plusieurs instructions machines, parfois des milliers, afin de rendre cette instruction compatible avec les différentes utilisations de celle-ci par le programmeur. La compilation (effectuée par un logiciel appelé compilateur) consiste à traduire un programme haut niveau en langage machine.

## 2. MODES D'ADRESSAGE

En langage assembleur, les modes d'adressage sont les manières d'accéder aux ressources du microprocesseur (ou microcontrôleur selon le cas). À chaque opérande (DST et SRC) doit être associé un mode d'adressage afin d'accéder à la ressource souhaitée.

### 2.1. ADRESSAGE DE REGISTRE

Certaines instructions sont spécifiques à certains registres. La ressource utilisée est alors contenue dans le code opératoire de l'instruction. Pour l'AT89C51, seuls les registres ACC, R0 à R7, DPTR (DPH et DPL) et l'adresse bit C sont concernés par ce mode d'adressage.

En langage assembleur, ce mode d'adressage est utilisé en notant directement le nom du registre comme opérande (sauf pour l'accumulateur qui est simplement noté A). Le tableau 1 donne l'exemple d'un adressage de registre en SRC et en DST, respectivement R0 et ACC. Cette instruction recopie la valeur de R0 dans ACC : 30h, comme illustré en figure 1.

---

1. Voir chapitre Pseudo-codes, page 41.

Pseudo-code <sup>1</sup>	Assembleur
(ACC) ← (R0)	MOV A , R0

TABLEAU 1 – Exemple d'adressage de registre



FIGURE 1 – Illustration de l'adressage de registre

## 2.2. ADRESSAGE IMMÉDIAT

L'adressage immédiat n'est pas réellement un adressage puisqu'il n'y a pas d'accès à une ressource. En effet, l'opérande est constituée de la donnée elle-même (ce mode d'adressage ne peut donc être qu'en source). En langage assembleur, ce mode d'adressage est symbolisé par un # qui précède le nombre. Ce dernier peut être représenté en décimal, il n'est suivi alors par aucune marque distinctive ; en hexadécimal, il est alors suivi de la lettre h ; ou en binaire, il est alors suivi de la lettre b.

Le tableau 2 donne l'exemple d'un adressage immédiat en SRC et de registre en DST, registre ACC. Cette instruction met la valeur 7Fh (127 en décimal) dans ACC, comme illustré en figure 2.

Pseudo-code	Assembleur
(ACC) ← 7Fh	MOV A , #7Fh

TABLEAU 2 – Exemple d'adressage immédiat



FIGURE 2 – Illustration de l'adressage immédiat

## 2.3. ADRESSAGE DIRECT

Dans l'adressage direct, l'opérande correspond à une adresse (sur 8 bits pour l'AT89C51) où se situe la donnée (à écrire ou à lire). Cette adresse peut correspondre à un registre SFR ou une adresse de la mémoire de données. En langage assembleur, ce mode d'adressage est directement symbolisé par l'adresse (sans signe distinctif).

Le tableau 3 donne l'exemple d'un adressage immédiat en SRC et en DST, avec deux adresses de la mémoire de données. Cette instruction, illustrée en figure 3, copie la valeur de la case mémoire de données d'adresse 2Fh dans la case mémoire de données d'adresse 30h.

Pseudo-code	Assembleur
$(30h) \leftarrow (2Fh)$	MOV 30h , 2Fh

TABLEAU 3 – Exemple d'adressage direct



FIGURE 3 – Illustration de l'adressage direct

## 2.4. ADRESSAGE INDIRECT

Dans l'adressage indirect, l'opérande correspond à un registre qui contient l'adresse où se situe la donnée. Pour l'AT89C51, seuls les registres R0, R1 et SP permettent de pointer une adresse de 8 bits (registre ou case de la mémoire de données) ; et le DPTR pour une adresse de 16 bits (mémoire programme). En langage assembleur, ce mode d'adressage est symbolisé par un @ qui précède l'adresse.

Le tableau 4 donne l'exemple d'un adressage indirect en SRC, registre R0, et de registre en DST, registre ACC. Cette instruction, illustrée en figure 4, met la valeur contenue dans la case mémoire pointée par R0 dans le registre ACC : 03h.

Pseudo-code	Assembleur
$(ACC) \leftarrow ((R0))$	MOV A , @R0

TABLEAU 4 – Exemple d'adressage indirect

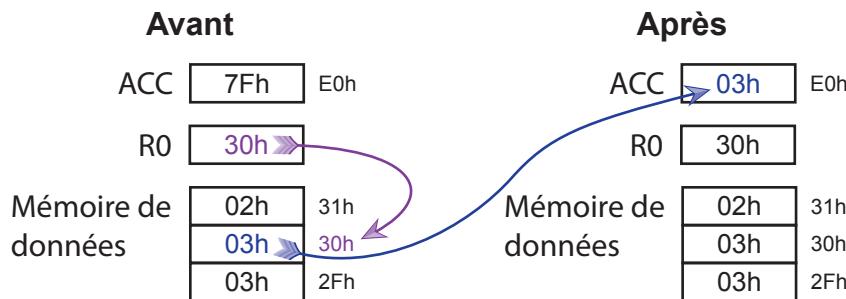


FIGURE 4 – Illustration de l'adressage indirect

## 2.5. ADRESSAGE INDEXÉ

L'adressage indexé permet de lire la mémoire programme, afin d'accéder à une table de correspondance (*lookup table*). L'adresse de la donnée est formée de la somme des valeurs d'un registre 16 bits (DPTR ou PC) et de l'accumulateur (ACC), le premier pointant la première adresse de la table et le second servant de décalage (numéro d'entrée de la table). Il s'agit d'une forme d'adressage indirect, d'où la présence du signe @ précédent l'appel à l'accumulateur (noté A). En langage assembleur, l'accès à la mémoire programme du microcontrôleur se fait à l'aide du mnémonique MOVC et une mémoire externe avec MOVX.

Le tableau 5 donne l'exemple d'un adressage indexé en SRC et de registre en DST. Cette instruction, illustrée en figure 5, met la valeur contenue dans la case mémoire pointée par la somme de ACC et DPTR dans le registre ACC : 03h.

Pseudo-code	Assembleur
$(ACC) \leftarrow ((ACC) + (DPTR))$	MOVC A, @A+DPTR

TABLEAU 5 – Exemple d'adressage indexé

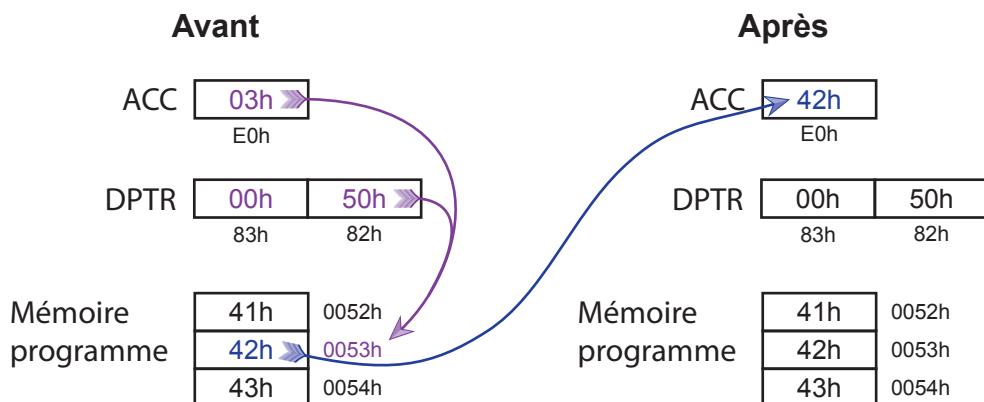


FIGURE 5 – Illustration de l'adressage indexé

Un autre type d'adressage indexé est utilisé dans l'instruction SJMP. Dans ce cas, l'adresse de destination d'une instruction de saut est calculée comme la somme du pointeur de base (DPTR) et de la valeur de l'accumulateur.

## 3. JEU D'INSTRUCTIONS

Une instruction assembleur est l'association d'un mnémonique et d'un mode d'adressage source et destination (la présence des SRC et DST dépend de l'instruction). C'est cet ensemble qui est associé à un code opératoire unique. Le CO ayant une taille de 1 octet, l'AT89C51 présente donc  $2^8 = 256$  combinaisons possibles (et pas une de plus), ce qui signifie que certaines combinaisons de modes d'adresses SRC et DST ne sont pas possibles.

La taille de l'instruction assembleur traduite en langage machine est fixe et exprimée en octet, cette information permet de prédire la taille d'un programme en mémoire programme. Sa durée est aussi constante et exprimée en nombre de cycle machine (CM) où 1 CM correspond à 12 cycles d'horloge (imposé par le quartz) :

$$T_{CM} = 12 / f_{OSC}$$

Certaines instructions modifient le registre d'état (PSW), le tableau 6 donne ces instructions et les bits du PSW modifiés.

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

TABLEAU 6 – Instructions qui modifient le PSW

Les instructions peuvent être classées en cinq catégories :

- instructions arithmétiques (manipulation d'octets) ;
- instructions logiques (manipulation d'octets) ;
- instructions de transfert de données (manipulation d'octets) ;
- instructions booléennes (manipulation de bits) ;
- instructions de saut de programme ;

Le tableau 7 donne le jeu d'instructions où les modes d'adressage sont repérés par :

- Rn ⇒ adressage de registre des registres R7 à R0 de la banque sélectionnée ;
- direct ⇒ adressage direct d'un registre SFR ou d'une case de la mémoire de donnée interne (adresse sur 8 bits) ;
- @Ri ⇒ adressage indirect (R0 ou R1) ;
- #data ⇒ adressage immédiat avec une constante sur 8 bits ;
- #data16 ⇒ adressage immédiat avec une constante sur 16 bits ;
- addr16 ⇒ adresse de destination sur 16 bits. Utilisé par les instructions LCALL et LJMP ; ces sauts permettent d'atteindre toute la mémoire programme.
- addr11 ⇒ adresse de destination sur 11 bits. Utilisé par les instructions ACALL et AJMP ; le saut doit se situer dans la même page de 2 Kio de la mémoire programme que l'adresse de l'instruction suivante (le PC fournit alors les 5 bits de poids fort de l'adresse) ;
- rel ⇒ décalage signé (complément à 2) sur une octet. Utilisé par SJMP et tous les sauts conditionnels, elle permet un saut relatif de -128 à +127 octets par rapport à l'adresse de l'instruction suivante ;
- bit ⇒ adressage direct d'une adresse bit (SFR ou mémoire de données interne).

Instruction	Description	Taille	Durée	CO
<b>INSTRUCTIONS ARITHMÉTIQUES</b>				
ADD A,Rn	(A) $\leftarrow$ (A) + (Rn)	1	1	28 à 2F
ADD A,direct	(A) $\leftarrow$ (A) + (direct)	2	1	25
ADD A,@Ri	(A) $\leftarrow$ (A) + ((Ri))	1	1	25, 26
ADD A,#data	(A) $\leftarrow$ (A) + data	2	1	24
ADDC A,Rn	(A) $\leftarrow$ (A) + (C) + (Rn)	1	1	38 à 3F
ADDC A,direct	(A) $\leftarrow$ (A) + (C) + (direct)	2	1	35
ADDC A,@Ri	(A) $\leftarrow$ (A) + (C) + ((Ri))	1	1	36, 37
ADDC A,#data	(A) $\leftarrow$ (A) + (C) + data	2	1	34
SUBB A,Rn	(A) $\leftarrow$ (A) - (C) - (Rn)	1	1	98 à 9F
SUBB A,direct	(A) $\leftarrow$ (A) - (C) - (direct)	2	1	95
SUBB A,@Ri	(A) $\leftarrow$ (A) - (C) - ((Ri))	1	1	96, 97
SUBB A,#data	(A) $\leftarrow$ (A) - (C) - data	2	1	94
INC A	(A) $\leftarrow$ (A) + 1	1	2	04
INC Rn	(Rn) $\leftarrow$ (Rn) + 1	1	1	08 à 0F
INC direct	(direct) $\leftarrow$ (direct) + 1	2	1	05
INC @Ri	((Ri)) $\leftarrow$ ((Ri)) + 1	1	1	06, 07
DEC A	(A) $\leftarrow$ (A) - 1	1	1	14
DEC Rn	(Rn) $\leftarrow$ (Rn) - 1	1	1	18 à 1F
DEC direct	(direct) $\leftarrow$ (direct) - 1	2	1	15
DEC @Ri	((Ri)) $\leftarrow$ ((Ri)) - 1	1	1	16, 17
INC DPTR	(DPTR) $\leftarrow$ (DPTR) + 1	1	2	A3
MUL AB	(A) <sub>7-0</sub> $\leftarrow$ (A) $\times$ (B) (B) <sub>15-8</sub>	1	4	A4
DIV AB	(A) $\leftarrow$ quotient de (A) / (B) (B) $\leftarrow$ reste de (A) / (B)	1	4	84
DA A	Ajustement décimal de A	1	1	D4
<b>INSTRUCTIONS LOGIQUES</b>				
ANL A,Rn	(A) $\leftarrow$ (A) ET (Rn)	1	1	58 à 5F
ANL A,direct	(A) $\leftarrow$ (A) ET (direct)	2	1	55
ANL A,@Ri	(A) $\leftarrow$ (A) ET ((Ri))	1	1	56, 57
ANL A,#data	(A) $\leftarrow$ (A) ET data	2	1	54
ANL direct,A	(direct) $\leftarrow$ (direct) ET (A)	2	1	52
ANL direct,#data	(direct) $\leftarrow$ (direct) ET data	3	2	53
ORL A,Rn	(A) $\leftarrow$ (A) OU (Rn)	1	1	48 à 4F
ORL A,direct	(A) $\leftarrow$ (A) OU (direct)	2	1	45
ORL A,@Ri	(A) $\leftarrow$ (A) OU ((Ri))	1	1	45, 47
ORL A,#data	(A) $\leftarrow$ (A) OU data	2	1	44
ORL direct,A	(direct) $\leftarrow$ (direct) OU (A)	2	1	42
ORL direct,#data	(direct) $\leftarrow$ (direct) OU data	3	2	43
XRL A,Rn	(A) $\leftarrow$ (A) OU EXCLUSIF (Rn)	1	1	68 à 6F
XRL A,direct	(A) $\leftarrow$ (A) OU EXCLUSIF (direct)	2	1	65
XRL A,@Ri	(A) $\leftarrow$ (A) OU EXCLUSIF ((Ri))	1	1	66, 67
XRL A,#data	(A) $\leftarrow$ (A) OU EXCLUSIF data	2	1	64
XRL direct,A	(direct) $\leftarrow$ (direct) OU EXCLUSIF (A)	2	1	62
XRL direct,#data	(direct) $\leftarrow$ (direct) OU EXCLUSIF data	3	2	63
CLR A	(A) $\leftarrow$ 0	1	1	E4
CPL A	(A) $\leftarrow$ $\overline{(A)}$	1	1	F4
RL A	Rotation à gauche de A	1	1	23
RLC A	Rotation à gauche avec la retenue (C) de A	1	1	33
RR A	Rotation à droite de A	1	1	03
RRC A	Rotation à droite avec la retenue (C) de A	1	1	13
SWAP A	(A <sub>3-0</sub> ) $\longleftrightarrow$ (A <sub>7-4</sub> )	1	1	CA

Instruction	Description	Taille	Durée	CO
<b>INSTRUCTIONS DE TRANSFERT DE DONNÉES</b>				
MOV A,Rn	(A) $\leftarrow$ (Rn)	1	1	E8 à EF
MOV A,direct	(A) $\leftarrow$ (direct)	2	1	E5
MOV A,@Ri	(A) $\leftarrow$ ((Ri))	1	1	E6, E7
MOV A,#data	(A) $\leftarrow$ data	2	1	74
MOV Rn,A	(Rn) $\leftarrow$ (A)	1	1	F8 à FF
MOV Rn,direct	(Rn) $\leftarrow$ (direct)	2	2	A8 à AF
MOV Rn,#data	(Rn) $\leftarrow$ data	2	1	78 à 7F
MOV direct,A	(direct) $\leftarrow$ (A)	2	1	F5
MOV direct,Rn	(direct) $\leftarrow$ (Rn)	2	2	88 à 8F
MOV direct <sub>1</sub> ,direct <sub>2</sub>	(direct <sub>1</sub> ) $\leftarrow$ (direct <sub>2</sub> )	3	2	85
MOV direct,@Ri	(direct) $\leftarrow$ ((Ri))	2	2	86, 87
MOV direct,#data	(direct) $\leftarrow$ data	3	2	75
MOV @Ri,A	((Ri)) $\leftarrow$ (A)	1	1	F6, F7
MOV @Ri,direct	((Ri)) $\leftarrow$ (direct)	2	2	A6, A7
MOV @Ri,#data	((Ri)) $\leftarrow$ data	2	1	76, 77
MOV DPTR,#data16	(DPTR) $\leftarrow$ data16	3	2	90
MOVC A,@A+DPTR	(A) $\leftarrow$ ((A) + (DPTR))	1	2	93
MOVC A,@A+PC	(A) $\leftarrow$ ((A) + (PC))	1	2	83
MOVX A,@Ri	Mémoire externe : (A) $\leftarrow$ ((Ri))	1	2	E2, E3
MOVX A,@DPTR	Mémoire externe : (A) $\leftarrow$ ((DPTR))	1	2	E0
MOVX @Ri,A	Mémoire externe : ((Ri)) $\leftarrow$ (A)	1	2	F2, F3
MOVX @DPTR,A	Mémoire externe : ((DPTR)) $\leftarrow$ (A)	1	2	F0
PUSH direct	(SP) $\leftarrow$ (SP) + 1 (SP) $\leftarrow$ (direct)	2	2	C0
POP direct	(direct) $\leftarrow$ ((SP)) (SP) $\leftarrow$ (SP) - 1	2	2	D0
XCH A,Rn	(A) $\longleftrightarrow$ (Rn)	1	1	C8 à CF
XCH A,direct	(A) $\longleftrightarrow$ (direct)	2	1	C5
XCH A,@Ri	(A) $\longleftrightarrow$ ((Ri))	1	1	C6, C7
XCHD A,@Ri	(A <sub>3-0</sub> ) $\longleftrightarrow$ ((Ri) <sub>3-0</sub> )	1	1	D6, D7

### INSTRUCTIONS BOOLÉENNES

CLR C	(C) $\leftarrow$ 0	1	1	C3
CLR bit	(bit) $\leftarrow$ 0	2	1	C2
SETB C	(C) $\leftarrow$ 1	1	1	D3
SETB bit	(bit) $\leftarrow$ 1	2	1	D2
CPL C	(C) $\leftarrow$ $\overline{(C)}$	1	1	B3
CPL bit	(bit) $\leftarrow$ $\overline{(bit)}$	2	1	B2
ANL C,bit	(C) $\leftarrow$ (C) ET (bit)	2	2	82
ANL C,/bit	(C) $\leftarrow$ (C) ET $\overline{(bit)}$	2	2	B0
ORL C,bit	(C) $\leftarrow$ (C) OU (bit)	2	2	72
ORL C,/bit	(C) $\leftarrow$ (C) OU $\overline{(bit)}$	2	2	A0
MOV C,bit	(C) $\leftarrow$ (bit)	2	1	A2
MOV bit,C	(bit) $\leftarrow$ (C)	2	2	92

Instruction	Description	Taille	Durée	CO
<b>INSTRUCTIONS DE SAUT</b>				
AJMP addr11	Saut absolue avec une adresse sur 11 bits	2	2	x1 (x pair)
LJMP addr16	Saut long : (PC) ← addr16	3	2	02
SJMP rel	Saut court : (PC) ← (PC) + rel	2	2	80
JMP @A+DPTR	Saut indirect : (PC) ← (A) + (DPTR)	1	2	73
JZ rel	Saut court si (A) = 0	2	2	60
JNZ rel	Saut court si (A) ≠ 0	2	2	70
CJNE A,direct,rel	Saut court si (A) ≠ (direct)	3	2	B5
CJNE A,#data,rel	Saut court si (A) ≠ data	3	2	B4
CJNE Rn,#data,rel	Saut court si (Rn) ≠ data	3	2	B8 à BF
CJNE @Ri,#data,rel	Saut court si ((Ri)) ≠ data	3	2	B6, B7
DJNZ Rn,rel	(Rn) ← (Rn) – 1 et saut court si (Rn) ≠ 0	2	2	D8 à DF
DJNZ direct,rel	(direct) ← (direct) – 1 et saut court si (direct) ≠ 0	3	2	D5
JC rel	Saut court si (C) = 1	2	2	40
JNC rel	Saut court si (C) = 0	2	2	50
JB bit,rel	Saut court si (bit) = 1	3	2	20
JNB bit,rel	Saut court si (bit) = 0	3	2	30
JBC bit,rel	Saut court et (bit) ← 0 si (bit) = 1	3	2	10
ACALL addr11	Appel absolue à une routine avec une adresse sur 11 bits	2	2	x1 (x impair)
LCALL addr16	Appel long à une routine	3	2	12
RET	Retour de routine	1	2	22
RETI	Retour de routine d'interruption	1	2	32
NOP	Pas d'opération	1	1	00

TABLEAU 7 – Jeu d'instructions de l'AT89C51

## 4. STRUCTURES EN ASSEMBLEUR

### 4.1. PROGRAMME PRINCIPAL

Comme illustré en figure 6, la structure d'un programme en langage assembleur doit contenir au moins les éléments suivants :

1. Un en-tête, sous forme de commentaire, qui indique le nom du programme, une description succincte, l'auteur, la version, la date, etc.
2. Une partie déclaration des symboles. Afin de rendre plus lisible le programme et faciliter sa maintenance, il est utile de nommer les constantes, les registres, les adresses de la mémoire de données utilisées.
3. Afin de réaliser une réinitialisation du microcontrôleur (*reset*) à la mise sous tension, un dispositif électronique (circuit RC) est branché sur la broche RST car c'est la seule méthode de « maîtriser » son état. À l'issue du *reset*, le PC vaut 0000h, c'est donc à cette adresse de la mémoire programme que doit se trouver la première instruction du programme, ce qui est réalisé avec la directive d'assemblage<sup>1</sup> : org 0000h.

1. Voir chapitre Ride, page 45.

4. Il s'ensuit le corps du programme principal qui doit être encadré par deux étiquettes (uniques) : *début* et *fin*, par exemple. La dernière instruction (au niveau de l'étiquette *fin*) doit être une instruction de saut inconditionnel vers une de ces étiquettes — lors de l'assemblage, l'opérande de l'instruction (l'adresse ou le décalage) est automatiquement calculée par l'assembleur). Sans cette instruction, le microcontrôleur exécuterait des instructions non souhaitées. En effet, le PC pointe l'adresse de l'instruction à exécuter et donc une case de la mémoire programme, avec une valeur entre 00h et FFh. Cette dernière correspond à un CO qui est exécuté par le microcontrôleur, le PC pointant alors « l'instruction » suivante, etc.

5. La directive d'assemblage *end* qui permet d'indiquer la fin du fichier à assembler.

*; En-tête*

```
; Déclaration des symboles
ALPHA          equ 10           ; constante
RES            data 30h         ; zone mémoire

; Implantation en mémoire programme
                    org 0000h

; Programme principal
début:
fin:
        end
```

FIGURE 6 – Structure minimum d'un programme

## 4.2. STRUCTURES SÉLECTIVES

La structure sélective, illustrée en tableau 8, est composée :

- d'étiquettes permettant de repérer les différentes parties de la structure ;
- d'un saut conditionnel vers la fin de la structure, avec une condition inversée par rapport à la condition en pseudo-codes ;
- d'une partie *Sinon* qui est facultative. En cas de présence, la dernière instruction de la partie *Si* doit être un saut inconditionnel vers la fin du *Sinon*.

Pseudo-code	Assembleur
<b>si</b> (I) = 0 <b>alors</b> : <b>sinon</b> : <b>fin</b>	SI : JNZ SINON FSI : LJMP FSINON SINON : FSINON :

TABLEAU 8 – Exemple de structure sélective en langage assembleur

### 4.3. STRUCTURES ITÉRATIVES

Les structures itératives *Répéter* et *Tant que*, illustrées en tableaux 9 et 10, sont composées :

- d'étiquettes permettant de repérer le début et la fin de la structure<sup>2</sup> ;
- d'un saut conditionnel avec une condition inversée par rapport à la condition en pseudo-codes ;
- dans le cas de la *tant que*, d'un saut inconditionnel vers le début de la structure.

Dans la plupart des cas, une instruction au sein de la structure doit modifier l'opérande de la condition, sous peine d'obtenir une boucle infinie.

Pseudo-code	Assembleur
$(l) \leftarrow 10$ <b>répéter</b> $\vdots$ $(l) \leftarrow (l) - 1$ $\vdots$ <b>jusqu'à</b> $(l) = 0$	$MOV A, \#10$ RPT : $DEC A$ JSQ :      JNZ RPT

TABLEAU 9 – Exemple d'une structure *Répéter* en langage assembleur

Pseudo-code	Assembleur
$(l) \leftarrow 10$ <b>tant que</b> $(l) \neq 0$ <b>faire</b> $\vdots$ $(l) \leftarrow (l) - 1$ $\vdots$ <b>fin</b>	$MOV A, \#10$ TQ :      JZ FTQ $DEC A$ SJMP TQ FTQ :

TABLEAU 10 – Exemple d'une structure *Tant que* en langage assembleur

### 4.4. ROUTINE

Une routine (ou sous-programme) est appelée à l'aide de l'instruction LCALL (ou ACALL) suivie du nom de l'étiquette qui repère le début de la routine — lors de l'assemblage, l'opérande de l'instruction (l'adresse) est automatiquement calculée par l'assembleur. Les opérations effectuées par l'instruction LCALL sont données en figure 7. Afin de revenir au programme appelant, la dernière instruction d'un sous-programme doit être RET, dont les opérations sont illustrées en figure 8.

2. Attention, une étiquette doit être unique dans le programme. De plus, une étiquette non utilisée n'a pas de conséquence sur le programme, ni en terme de place mémoire, ni en terme de temps d'exécution puisqu'elle est remplacée par son adresse ou le décalage lors de l'assemblage.

```

(PC) ← (PC) + 3           // taille de l'instruction LCALL
(SP) ← (SP) + 1           // sauvegarde du PC dans la pile
((SP)) ← (PC7-0)        // bits 0 à 7 du PC
(SP) ← (SP) + 1
((SP)) ← (PC15-8)       // bits 8 à 15 du PC
(PC) ← addr16              // saut au sous-programme

```

FIGURE 7 – Opérations réalisées par l'instruction LCALL

```

(PC15-8) ← ((SP))    // restauration de la pile dans le PC
(SP) ← (SP) - 1
(PC7-0) ← ((SP))
(SP) ← (SP) - 1

```

FIGURE 8 – Opérations réalisées par l'instruction RET

Hormis pour les paramètres retournés (et éventuellement passés), une routine ne doit pas modifier la valeur des registres et de la mémoire de données. Les éléments utilisés doivent donc être sauvegardés en début de routine et restaurés en fin de routine dans la pile à l'aide des instructions, respectivement, PUSH et POP. La pile étant de type LIFO, l'ordre de restauration doit être inversé par rapport à la sauvegarde dans la pile. Il est utile de placer un en-tête avant la première instruction de chaque routine afin d'indiquer une description succincte, les paramètres passés et retournés, etc.

#### 4.5. ROUTINE D'INTERRUPTION

En regardant la table des vecteurs d'interruption (cf. tableau 5 page 22), on peut remarquer que deux vecteurs (adresse d'IT) sont espacés de 8 octets (hormis pour le *reset* qui ne dispose que de 3 octets), ce qui laisse peu de place pour une routine d'interruption. Si l'on souhaite disposer de plus d'espace mémoire sans empiéter sur un autre vecteur, il suffit de faire un saut inconditionnel, à l'adresse du vecteur, vers une étiquette placée plus loin en mémoire programme (l'instruction LJMP a une taille de 3 octets), comme le montre l'exemple donné en figure 9.

Une routine d'interruption doit se terminer avec l'instruction RETI. Son fonctionnement est similaire à l'instruction RET mais qui, en plus, restaure le niveau de priorité antérieur à l'appel de l'IT en question.

```

;En-tête

;Déclaration des symboles

; Table des vecteurs d'IT
org 0000h
SJMP 0030h

org 000Bh ; IT Timer 0
CLR C ; Commentaires
RETI

org 0013h ; IT externe 1
LJMP IT_IE1

org 00030h ; saut table des vecteurs

; Programme principal
début:

fin: LJMP début

; Routine d'IT INTO \
; Description de la routine
; Paramètres entrants et sortants
IT_IE1:

Fin_IT_IE1: RETI

end

```

FIGURE 9 – Exemple de structure d'une routine d'interruption

# ANALYSE STRUCTURÉE

## 1. INTRODUCTION

Une bonne analyse du problème est un préambule indispensable avant toute tentative de programmation. En effet, il est illusoire d'obtenir un programme structuré, efficace et valide en se lançant tête baissée dans l'écriture de lignes de codes sans avoir bien cerné le problème posé et la solution proposée avec ses tenants et aboutissants. Il est donc nécessaire de procéder par étape :

1. Analyse descendante (ou montante).
2. Arbre programmatique.
3. Tableau description des données.
4. Traduction en pseudo-codes.
5. Traduction dans le langage choisi (langage assembleur, langage C, etc.).
6. Assembler ou compiler le programme.
7. Dans le cas d'un circuit intégré programmable (microcontrôleur par exemple), le programmer.

Bien entendu, selon la complexité du problème et l'envergure du programme qui en découle, certaines étapes ne seront pas obligatoires (étape 2 par exemple). De plus, il ne faut pas penser que cela représente un modèle linéaire dans le temps, mais qu'il est parfois nécessaire d'effectuer des aller-retours entre les différentes étapes (oublie d'un cas de figure, contrainte matérielle, etc.).

La suite va détailler ces différentes étapes qui seront illustrées à l'aide de l'exemple suivant : « écrire, en langage assembleur pour AT89C51, un sous-programme du programme principal *Bingo* qui réalise la permutation de la valeur de trois variables *X*, *Y* et *Z* (entiers), telle qu'illustrée en figure1 ».

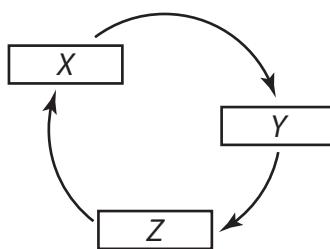


FIGURE 1 – Permutation de trois valeurs

## 2. ANALYSE DESCENDANTE

L'analyse descendante consiste à partir du problème initial et à le décomposer en sous-problème de plus en plus simple à résoudre.

Dans notre exemple, le sous-programme demandé doit réaliser une tâche simple, il n'est donc pas nécessaire de le décomposer en sous-programme<sup>1</sup>.

1. Le fait de dire qu'une chose n'est pas réalisée fait partie intégrante d'une analyse structurée puisqu'ainsi, aucune option n'est oubliée.

Il reste donc à décomposer le problème en étape élémentaire :

1. Mettre la valeur de  $X$  dans  $Y$ .
2. Mettre la valeur de  $Y$  dans  $Z$ .
3. Mettre la valeur de  $Z$  dans  $X$ .

Si on teste cette algorithme avec des valeurs numériques ( $X = 1$ ,  $Y = 2$  et  $Z = 3$  par exemple), on se rendrait compte, qu'à la fin, toutes les variables contiennent la valeur de  $X$ . En effet, la valeur de  $Y$  est écrasée lors de l'étape 1 ; cela montre l'intérêt de tester son algorithme sur un exemple simple.

Il faut donc utiliser une variable supplémentaire qui sera appeler  $TEMP$  (pour temporaire) :

1. Mettre la valeur de  $X$  dans  $TEMP$ .
2. Mettre la valeur de  $Z$  dans  $X$ .
3. Mettre la valeur de  $Y$  dans  $Z$ .
4. Mettre la valeur de  $TEMP$  dans  $Y$ .

Cette fois-ci, la vérification avec des valeurs montre que l'algorithme est valide.

### 3. ARBRE PROGRAMMATIQUE

Un arbre programme représente l'ensemble des sous-programmes appelés par un programme principal et se lit toujours du haut vers le bas et de la droite vers la gauche. Dans cette représentation (voir figure 2), le programme principale se trouve donc tout en haut et s'en suit les différentes hiérarchies de sous-programmes, ces derniers étant représentés autant de fois qu'ils sont appelés. Chacun est représenté par un rectangle dont la partie supérieure mentionne le nom du (sous-)programme ; la partie inférieure peut comporter une description succincte ; à gauche et à droite sont reportés les paramètres respectivement entrants et sortants.

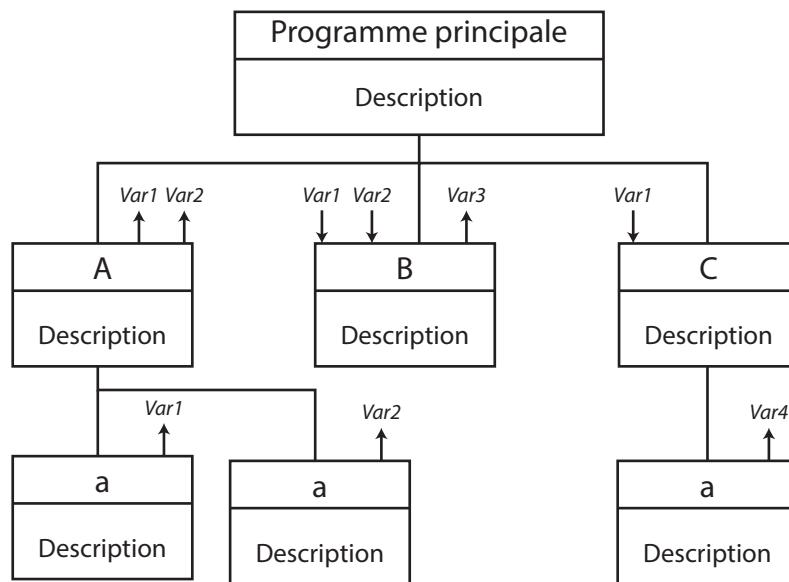


FIGURE 2 – Exemple d'arbre programmatique

Dans l'exemple, il y a un programme principal nommé Bingo et le sous-programme à écrire qui sera nommé Permute\_3 (il faut toujours donner des noms parlants) qui a trois paramètres entrants ( $X$ ,  $Y$  et  $Z$ ) et ces mêmes paramètres sortants (car elles sont modifiées par le sous-programme). Comme le montre la figure 3, la variable  $TEMP$  n'apparaît pas car elle est interne au sous-programme.

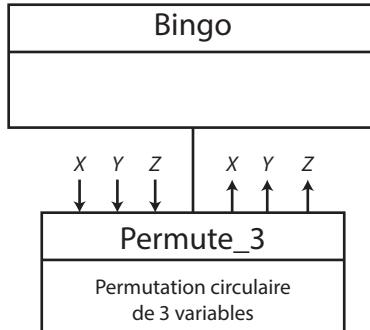


FIGURE 3 – Arbre programmatique de l'exemple

#### 4. TABLEAU DESCRIPTION DES DONNÉES

Le tableau de description des données rassemble l'ensemble des variables manipulées avec leur type (entier, réel, booléen, chaîne de caractère, etc.), une brève description et la ressource matérielle utilisée, dans le cas d'une programmation bas niveau.

Pour notre exemple, il y a quatre variables :  $X$ ,  $Y$ ,  $Z$  et  $TEMP$  qui sont toutes de type entier. Les trois premières seront stockées dans les registres  $R0$ ,  $R1$  et  $R2$  et, l'accumulateur peut être un choix judicieux pour  $TEMP$  qui est une variable temporaire. Le résultat est donné par le tableau 1.

Nom	Type	Description	Ressource
$X$	Entier	variable à permuter	$R0$
$Y$	Entier	variable à permuter	$R1$
$Z$	Entier	variable à permuter	$R2$
$TEMP$	Entier	variable temporaire	ACC

TABLEAU 1 – Tableau de description des données de l'exemple

#### 5. PSEUDO-CODES

Comme leur nom l'indique, les pseudo-codes ne sont pas un langage de programmation mais une manière de représenter un algorithme en suite d'action élémentaire de manière linéaire et à une dimension (à l'inverse des organigramme). Indépendant du matériel et du langage de programmation, les pseudos-codes peuvent être vues comme une étape de transition entre l'analyse et l'écriture du programme. N'étant pas un langage, il ne faut pas chercher une rigidité dans son écriture même si, certaines « règles » sont données ci-après ; par exemple, il est tout à fait possible d'utiliser des phrases en langage naturel.

## 5.1. STRUCTURE D'UN PROGRAMME

Un (sous-)programme doit obligatoirement comporter un et un seul point d'entrée, repéré par son nom et suivi par le mot **début**, et un et un seul point de sortie, repéré par le mot **fin**. En effet, l'emploi d'instruction de type GOTO va à l'encontre de la notion de programme structuré.

Afin de faciliter la lecture, des indentations sont employées pour repérer les différents niveaux de structure (le (sous-)programme représente le premier niveau).

## 5.2. VARIABLE

Les variables devront être désignées avec un nom significatif (éviter l'emploi de *A*, *B*, etc. même si l'exemple ne suit pas cette règle).

Selon la programmation envisagée, langage de bas ou de haut niveau, les objets manipulés ne sont pas représentés de la même manière. En effet, lorsqu'on programme avec un langage bas niveau (langage assembleur), il est important de connaître les modes d'adressage et l'emploi de parenthèse pour parler du contenu de la variable peut être intéressant ; alors qu'avec un langage haut niveau, une telle question ne se pose pas. Ainsi les notations (*X*) et *X* peuvent être vues comme similaires. Par la suite, la première notation sera retenue.

## 5.3. AFFECTATION

L'opération d'affectation consiste à initialiser une variable à avec une valeur ; cette dernière peut être suivie d'un *h* ou d'un *b* dans le cas d'une donnée codée en hexadécimal ou en binaire. Le symbole associé à cette opération est :  $\leftarrow$  (le signe égal est utilisé pour l'opération de comparaison).

Par exemple (*X*)  $\leftarrow$  10 signifie qu'après cette opération, la variable *X* contiendra la valeur 10.

## 5.4. OPÉRATIONS

Toutes les opérations possibles peuvent être représentées avec leur symbole usuel. Pour des opérations particulières qui ne possède pas de symbole, reste au programmeur à décrire sous forme de mots clairs ou d'en inventer un.

Par exemple, l'opération (*X*)  $\leftarrow$  *X* + 1, permet d'incrémenter le contenu de la variable *X* (si *X* avait pour valeur 10 avant cette opération, elle contiendra 11 après).

## 5.5. STRUCTURE SÉLECTIVE

Une structure sélective permet de réaliser des opérations lorsqu'une condition est réalisée (*si condition alors*) et, éventuellement, d'autres dans le cas contraire (*sinon*), voir figure 2.

## 5.6. STRUCTURES ITÉRATIVES

Une structure itérative permet de réaliser des opérations plusieurs fois, selon une condition particulière. Les deux structures itératives de base sont la *Répéter* et la *Tant que* (voir tableau 2). La différence entre ces deux structures réside dans le fait que la **Répéter** est exécuté au moins une fois puisque la condition est vérifiée à la fin ; ce qui n'est pas forcément le cas de la **Tant que** qui a le test de la condition au début. Dans les deux cas, il est impératif d'avoir une opération qui modifie la ou les variables testées dans la condition sous peine d'avoir une boucle infinie.

Structure sélective		Structures itératives
<b>si</b> ( $X$ ) = 0 <b>alors</b> : <b>sinon</b> : <b>fin</b>	<b>répéter</b> : $(X) \leftarrow (X) - 1$ : <b>jusqu'à</b> ( $X$ ) = 0	<b>tant que</b> ( $X$ ) $\neq$ 0 <b>faire</b> : $(X) \leftarrow (X) - 1$ : <b>fin</b>

TABLEAU 2 – Exemple de structures itérative et sélectives

### 5.7. APPEL À UN SOUS-PROGRAMME

L'appel à un sous programme (ou routine) s'effectue par son nom suivi de parenthèses. Les éventuels paramètres passés et retournés sont listés (séparés par des virgules) réciproquement entre les parenthèses et à gauche d'un opérateur d'affectation.

### 5.8. EXEMPLE

Dans l'exemple pris, les pseudo-codes sont donnés par le tableau 3.

```

Permute_3
début
  ( $TEMP$ )  $\leftarrow$  ( $X$ )
  ( $X$ )  $\leftarrow$  ( $Z$ )
  ( $Y$ )  $\leftarrow$  ( $Y$ )
  ( $Z$ )  $\leftarrow$  ( $TEMP$ )
fin

```

TABLEAU 3 – Pseudo-codes de l'exemple

## 6. LANGAGE ASSEMBLEUR AT89C51

Le passage des pseudo-codes en langage assembleur semble ais  puisqu'un seul type d'op ration intervient, l'affectation, qui correspond au mn monique MOV. Les variables  $X$ ,  $Y$  et  $Z$  sont dans les registres R0 ¦ R3, or, en regardant de plus pr s les combinaisons de modes d'adressage possibles (voir jeux d'instruction page 34), on peut se rendre compte qu'il n'est pas possible d'avoir de l'adressage de registre Rn en source et en destination (l'instruction MOV Rn,RN n'existe pas). Il est donc n cessaire de passer par un interm diaire et l'accumulateur semble tout d sign  (taille de 1 octet et dur e de 1 cycle machine). Cela implique de changer la ressource mat rielle pour la variable TEMP qui peut  tre le registre B. Le nouveau tableau de description des donn es est alors celui pr sent  en tableau 4 et le programme en langage assembleur est donn  en tableau 5. On pourra remarquer que cela n'affecte en rien les pseudo-codes (c'est bien l  leur avantage : ind pendance du mat riel et du langage de programmation).

Nom	Type	Description	Ressource
X	Entier	variable à permuter	R0
Y	Entier	variable à permuter	R1
Z	Entier	variable à permuter	R2
TEMP	Entier	variable temporaire	ACC B

TABLEAU 4 – Tableau de description des données de l'exemple

Pseudo-code	Assembleur
<b>Permute_3</b> <b>début</b>  $(TEMP) \leftarrow (X)$ $(X) \leftarrow (Z)$ $(Z) \leftarrow (Y)$ $(Y) \leftarrow (TEMP)$  <b>fin</b>	<b>Permute_3 :</b> PUSH ACC ; Sauvegarde de A PUSH B ; et B dans la pile  MOV B, R0  MOV A, R2 MOV R0, A  MOV A, R1 MOV R2, A  MOV R1, B  POP B ; Restauration de B POP ACC ; et A depuis la pile <b>fin_p3:</b> RET ; retour sous-prog

TABLEAU 5 – Programme en langage assembleur de l'exemple

# RIDE

## 1. INSTALLATION

The Raisonance Integrated Development Environment (Ride) est une interface logicielle complète proposée par Raisonance pour l'écriture, la compilation et le débogage de programme, en langage assembleur ou en langage C pour nombre de microprocesseur dont l'AT89C51. Le fichier fourni (`RKit61.zip`) correspond à une version 6.10 d'évaluation. Celle-ci n'est pas la dernière en date mais a l'avantage d'être gratuite et complète pour l'utilisation qui va en être faite.

La procédure d'installation, sur PC Windows, est :

1. Décompresser le fichier `RKit61.zip` dans un répertoire (... \RKit61 par exemple).
2. Exécuter le fichier `INSTALL.EXE` du répertoire ... \RKit61\ EVAL\.
3. Poursuivre l'installation jusqu'à la fenêtre donnée en figure 1 (clics sur les boutons `NEXT` ou `ACCEPT`).
4. Sur la fenêtre *Type of Setup* (voir figure 1) :
  - cocher l'option *80C51 ToolChain*;
  - décocher l'option *RLink USB Drivers*;
  - choisir un répertoire d'installation (`C:\RIDE` par défaut);
  - cliquer sur le bouton `NEXT`.
5. Terminer l'installation (clic sur le bouton `NEXT`).
6. Ouvrir le répertoire d'installation précédemment défini (`C:\RIDE` par défaut).
7. Créer un raccourci sur le bureau du fichier `C:\RIDE\BIN\ride.exe` (fichier d'exécution du programme Ride).
8. Supprimer le répertoire de fichier compressé ... \RKit61\ EVAL\.

Pour pallier une incompatibilité relative de cette version de Ride avec les OS actuels, il est conseillé d'exécuter le programme en compatibilité Windows 95 (clic gauche sur le raccourci bureau ► Propriétés ► Compatibilité ; voir figure 2).

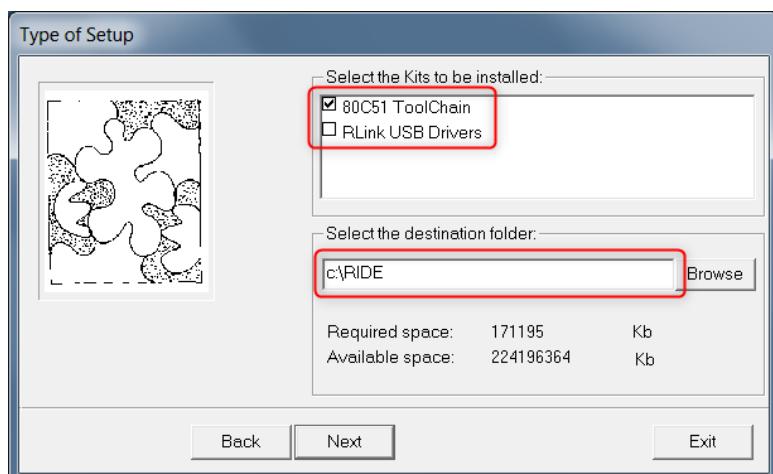


FIGURE 1 – Fenêtre de configuration de l'installation de Ride

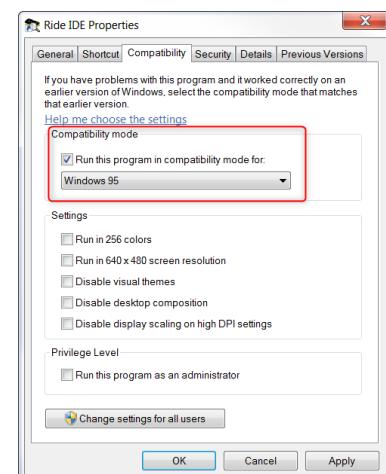


FIGURE 2 – Fenêtre *Propriété*, onglet *Compatibilité*

## 2. PROJET

Ride travaille à l'aide de fichiers inclus dans un projet :

1. Créer un projet à l'aide du menu Project ► New.
2. Dans la première fenêtre de configuration (voir figure 3) :
  - A. Renseigner le nom du projet dans *Application Name* (nom générique et pas obligatoirement celui du programme ; 8 caractères maximum et composé de caractères alphanumériques et *underscore*).
  - B. Choisir un répertoire dans le champ *Directory* (C:\PHY3101\Groupe\_X où X est l'identifiant de votre groupe).
  - C. Dans le champ *Target Family*, vérifier que le type de processeur est 80C51.
  - D. Laisser le champ *Type of Application* sur Application.
  - E. Appuyer sur le bouton NEXT.

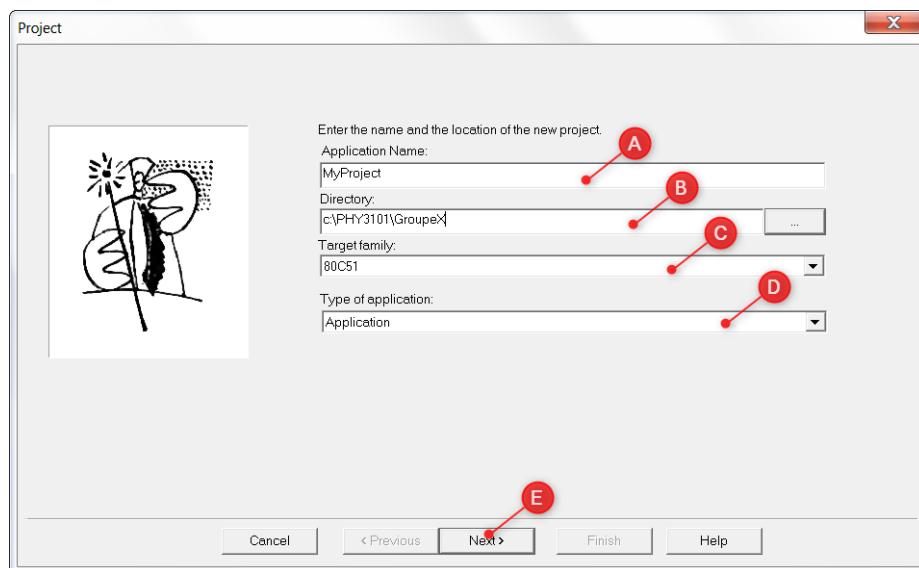


FIGURE 3 – Fenêtre de configuration d'un projet

3. Dans le fenêtre suivante, dans l'onglet *Device*, choisir la version du microcontrôleur : répertoire ATMEL (fabricant) — modèle AT89C51.
4. Terminer la configuration du projet en cliquant sur le bouton FINISH.

La gestion se fait à l'aide de la fenêtre *Project* dans laquelle doit apparaître le projet créé (extension NomProjet.AOF), avec son chemin. Si cette fenêtre n'apparait pas, sélectionner le menu View ► Project.

## 3. FICHIER

Un programme s'écrit à l'aide d'un éditeur de texte que propose Ride sous la forme de fichiers. Pour un programme écrit en langage assembleur :

1. Menu File ► New ► Assembler Files.
2. Apparition d'une fenêtre nommée *untitled.a51*, l'extension a51 correspondant au fichier en langage assembleur pour la famille des processeurs 80C51.

### 3. Sauvegarder le fichier par :

- **attention, ne pas faire** de Save qui correspond à la sauvegarde dans le répertoire par défaut de Ride qui est non connu ; mais
- utiliser le menu File ► Save As ;
- vérifier le chemin de sauvegarde (C:\PHY3101\Groupe\_X) dans le champ Save in ;
- dans le champ *File name*, renseigner un nom significatif (le nom du programme par exemple, suivi d'un numéro de version ; 8 caractères maximum et composé de caractères alphanumériques et underscore) ;
- vérifier le type du fichier : *Assembler Files* ;
- appuyer sur le bouton Save ;
- toute nouvelle sauvegarde pourra être faite avec le menu File ► Save (raccourci CTRL+S) ;
- associer le fichier au projet :
  - menu Projet ► Add node Source\_Application (ALT+INS) ;
  - sélectionner le fichier à associer dans le champ *File name* ;
  - cliquer sur le bouton Open ;
  - le fichier doit apparaître dans l'arborescence du projet (fenêtre *Project*) .

Il est possible de supprimer l'association d'un fichier avec un projet, ce qui n'a pas pour conséquence de le supprimer du disque dur :

1. À l'aide d'un clic gauche de la souris, sélectionner un fichier dans la fenêtre *Project* ;
2. Supprimer l'association par le menu Projet ► Delete node (Alt+Del).

## 4. PROGRAMME

### 4.1. COMMENTAIRE

Un commentaire peut être placé n'importe où ; il commence par un point-virgule et tout ce qui suit cette marque sera considéré comme un commentaire. L'éditeur de texte utilise un code couleur : les commentaires sont en vert, les mots réservés en noir et gras, etc.

### 4.2. CHAMPS

L'écriture en langage assembleur requiert trois champs : étiquette, mnémonique et opérande. Le passage d'un champ à un autre se fait à l'aide d'au moins un espace. Cependant, afin de faciliter la lecture d'un programme, il est important d'aligner chaque champ à l'aide de tabulations.

Une étiquette est un symbole utilisé pour marquer un endroit particulier dans un programme. Il peut faire référence, entre autre, à un code du programme (utilisé pour les instructions de saut, par exemple) ou à des données constantes stockées dans la mémoire programme. Toutes les étiquettes sont remplacées par leur adresse (ou le décalage, selon l'instruction) respective par l'assembleur pour celles utilisées par le programme, pour les autres, elles sont totalement transparentes au niveau du langage machine. Une **étiquette** doit être composée d'un **nom** de 20 caractères maximum (caractères alphanumériques, non sensible à la casse, et underscore et commençant par une lettre — attention aux mots réservés), **suivi de deux-points** (:).

Concernant les opérandes, il est plus simple d'utiliser les noms des SFR plutôt que leur adresse. Afin qu'ils soient reconnus par l'assembleur, il faut configurer le projet :

1. Menu *Options* ► *Project*.
2. Dans la fenêtre *Options*, développer la section *MA51*.
3. Dans l'option *Source*, valider *Define symbols for the 8051 function registers*.
4. Cliquer sur le bouton *OK*.

### 4.3. DIRECTIVE D'ASSEMBLAGE

Comme leur nom l'indique, les directives d'assemblage sont des commandes à destination de l'assembleur et sont donc totalement transparentes pour le microcontrôleur (elles ne sont pas traduites en langage machine). Elles sont composées d'un maximum de trois champs (à aligner, pour une lecture plus aisée, avec les champs du langage assembleur) :

- symbole qui suit la même règle typographique qu'une étiquette **sans** les deux-points. Le fait qu'ils doivent commencer par une lettre impose qu'un nombre doit commencer par un chiffre (les nombres en hexadécimal peuvent être précédés d'un zéro) ;
- directive d'assemblage ;
- expression.

Une liste exhaustive de ces directives peut être consultée sur la documentation de Ride<sup>1</sup>, dont en voici quelques unes :

- ORG addr ⇒ permet d'indiquer l'adresse en mémoire programme où se situe l'instruction qui suit cette directive (remplissage automatique des cases mémoires intermédiaires) ;
- END ⇒ spécifie la fin du fichier à assembler (tout ce qui suit ne sera pas pris en compte). Cette directive doit obligatoirement être placée à la fin du programme ;
- symb BIT bit-add ⇒ associe un symbole à une adresse bit de la mémoire de données ;
- symb DATA addr ⇒ associe un symbole à une adresse octet de la mémoire de données ;
- symb EQU exp ou symb EQU reg ⇒ associe un symbole à une valeur numérique ou à un registre (A ou R0-R7).

## 5. ASSEMBLAGE

La traduction du programme en langage assembleur en langage machine se fait à l'aide du menu *Project* ► *Build all* (Shift+F9). En bas de la fenêtre principale, la fenêtre *Make* indique le résultat de l'assemblage : symboles rouges en cas d'erreur et verts si l'assemblage est réussi.

En cas d'erreur, une explication est donnée dans la fenêtre *Make*, ainsi que sa position dans le programme (numéro de ligne). Il y a deux moyens d'accéder à une erreur :

- faire un clic droit dans la fenêtre du programme (éditeur de texte), choisir *Go to Line* (CTRL+G), renseigner le numéro de ligne souhaité dans le champ *Line* et cliquer sur le bouton *OK* ; ou
- parcourir le(s) erreur(s) par le menu *Search* ► *Next message* ou *Previous message* (respectivement ALT+F8 et ALT+F9).

---

1. Menu *Help* ► *PDF* ► *8051-Tools* ► *MA51 [6]*.

La réussite de l'assemblage indique qu'aucune erreur sémantique<sup>2</sup> n'est présente. En ouvrant l'arborescence du dernier symbole vert, Ride indique la génération du fichier en langage machine avec le nom du projet et l'extension HEX.

## 6. DÉBOGUEUR

Lors de l'assemblage, Ride effectue des vérifications sémantiques mais pas fonctionnelles<sup>3</sup>. Le mode débogage permet d'exécuter le programme pas à pas et de contrôler les valeurs des différents registres, ports, etc. par rapport à celles prédites par l'algorithme.

### 6.1. CONFIGURATION

Avant de l'utiliser, il faut configurer le débogueur (à faire à chaque nouveau projet) :

1. Ouvrir la fenêtre *Debug Options* à l'aide du menu *Options ▶ Debug*.
2. Dans la fenêtre *Debug Options* (voir figure 4) :
  - A. Sélectionner l'option *Virtual Machine (Simulator)*.
  - B. Vérifier que le champ *Device* indique AT89C51.
  - C. Régler la fréquence du quartz ( $f_{OSC} = 12$  MHz).
  - D. Cliquer sur le bouton *Advanced Options*.
3. Dans la fenêtre *Application options* (voir figure 4) :
  - A. Le champ *Code Size* correspond à la taille de la mémoire programme en kiboc-  
tets : 4 KiB pour l'AT89C51 (pas de mémoire programme externe).
  - B. Les champs *XData Size* et *XData Offset* sont à renseigner lorsqu'il y a une mé-  
moire de données externes : 0 KiB dans notre cas.
  - C. Laisser décocher l'option *Embedded ROM version*.
  - D. Cliquer sur le bouton *Ok*.
4. Cliquer sur le bouton *OK* de la fenêtre *Debug Options*.

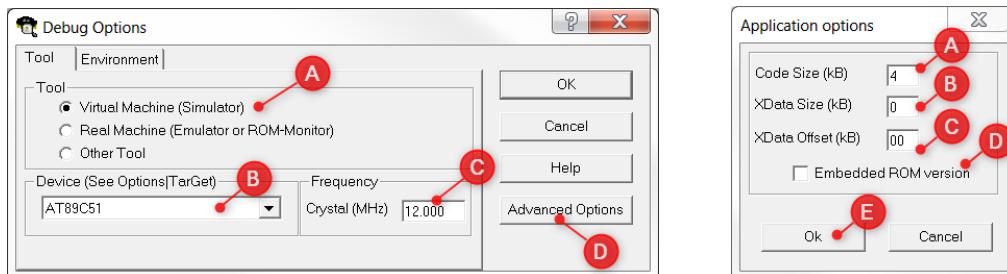


FIGURE 4 – Fenêtres de configuration du débogueur de Ride

2. Instruction assembleur qui ne peut pas être traduite en langage machine, par exemple : ADD B,A.  
 3. Par exemple, la traduction du pseudo-code (A) ← 10h par l'instruction assembleur MOV A,10h. Cette dernière est traduisible en langage machine mais elle ne fera pas ce qui est attendu puisqu'elle correspond au pseudo-code (A) ← (10h).

## 6.2. MODE DÉBOGAGE

Le passage en mode débogage s'effectue à l'aide du menu *Debug* ► *Start NomDuProjet.aof* (CTRL+D). En effet, c'est le projet qui est débogué et plus précisément le fichier HEX (en langage machine), ce qui veut dire que toute modification du fichier texte (.a51) n'est prise en compte qu'après arrêt du débogueur (menu *Debug* ► *Terminate* ou CTRL+SHIFT+D) et assemblage à nouveau du projet (SHIFT+F9). Il ne faut donc jamais modifier un programme en mode débogueur. Dans ce mode, la fenêtre principale est similaire à celle illustrée en figure 5, où l'on trouve :

- A. La barre d'outils du débogueur.
  - B. La fenêtre *Debugger* qui permet d'accéder aux différentes ressources matérielles du microcontrôleur.
  - C. La fenêtre Programme.
  - D. La fenêtre *Watches* qui permet de suivre l'évolution de variables.

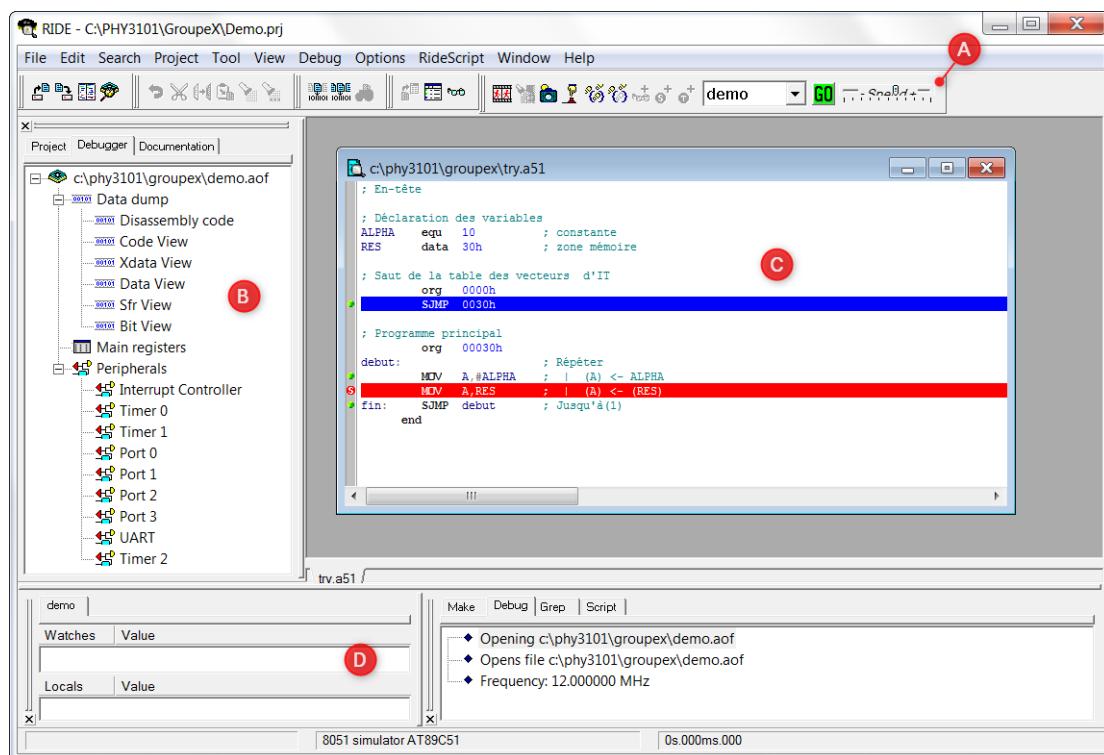


FIGURE 5 – Fenêtre de Ride en mode débogueur

## 6.3. BARRE D'OUTILS DU DÉBOGUEUR

Les outils disponibles dans la barre d'outils du débogueur sont (de la gauche vers la droite de la figure 5) :

-  *Animate* ⇒ mise à jour automatique des ressources matérielles lors de l'exécution du programme ;
  -  *Display executed line* ⇒ repérage des lignes du programme déjà exécutées par un point bleu dans la fenêtre Programme ;
  -  *Refresh* ⇒ mise à jour des ressources matérielles ;
  -  *Reset* ⇒ réalisation d'un *reset* ;

- *Step into* ⇒ exécution d'une instruction (raccourci clavier F7) ;
- *Step over* ⇒ exécution d'une instruction ou, le cas échéant, d'un sous-programme ;
- *Add watch* ⇒ ajout d'une variable dans la fenêtre *Watches* ;
- *Toggle Breakpoint* ⇒ ajout ou suppression d'un point d'arrêt sur la ligne sélectionnée de la fenêtre Programme ;
- *Toggle Trace* ⇒ ajout ou suppression d'une sonde sur la ligne sélectionnée de la fenêtre Programme ;
- *Run/Stop* ⇒ exécution/arrêt du programme en continu ;
- *Speed* ⇒ réglage de la vitesse d'exécution du programme en continu (il est conseillé de mettre, dans un premier temps, le curseur au niveau du S de *Speed* afin de visualiser les changements des ressources matérielles).

#### 6.4. FENÊTRE *Debugger*

Cette fenêtre permet de visualiser et modifier les ressources matérielles du microcontrôleur, son arborescence est composée de trois parties principales. Premièrement, la partie *Data dump* permet d'accéder aux différentes mémoires, dont :

- *Code View* ⇒ visualisation de la mémoire programme ;
- *Data View* ⇒ visualisation de la mémoire de données (adresses de 00h à FFh). Le champ *Search* permet d'accéder directement à une case mémoire souhaitée. Un double-clic sur une case mémoire permet de modifier sa valeur ;
- *Sfr View* ⇒ visualisation des registres de fonction spéciale qui sont repérés en magenta. Le champ *Search* permet d'accéder directement à un SFR en renseignant son nom (ACC pour l'accumulateur, par exemple). Un double-clic sur une case mémoire permet de modifier sa valeur (la mise à jour des ports peut n'être réalisée que lors de l'exécution de l'instruction suivante) ;
- *Bit View* ⇒ visualisation de la zone bit de la mémoire de données (adresses de 00h à 7Fh) et des bits des SFR accessibles par bit, ces derniers sont repérés en magenta. Le champ *Search* permet d'accéder directement à un bit à l'aide de son adresse ou de son nom pour les bits des SFR (ACC.0 pour le bit de poids faible de l'accumulateur, par exemple). Un double-clic sur un bit permet de modifier sa valeur (0 ou 1).

Deuxièmement, la partie *Main registers* peut être vue comme un « tableau de bord », puisqu'elle regroupe les principaux registres du microcontrôleur, avec entre autres (cf. figure 6) :

- le compteur ordinal (PC) ;
- l'accumulateur (ACC) et le registre B ;
- le registre d'état (PSW) et la valeur de son bit C (*Carry* ou retenue) ;
- la banque sélectionnée (RB) et la valeur des registres R0 à R7 associés, ainsi que les valeurs pointées par les registres R0 et R1 (@R0 et @R1) ;
- la valeur des registres P0 à P3 (valeur des ports en sortie, voir ci-dessous pour plus de détails).

CPU	Bank	Data	Hardware
PC 0000	RB 00	@R0 00	P0 FF
ACC 00	R0 00	@R1 00	P1 FF
PSW 00	R1 00	@DPTR FF	P2 FF
SP 07	R2 00	X@R0 FF	P3 FF
DPTR 0000	R3 00	X@R1 FF	TCON 00
B 00	R4 00	SPX XXX	THL0 0000
C 0	R5 00	BANK XXX	THL1 0000
EA 0	R6 00	Task XXX	THL2 0000
IE 00	R7 00	TaskP XXX	PCON 00

FIGURE 6 – Fenêtre *Main register*

La troisième partie nommée *Peripherals* permet de contrôler les différentes entrées/sorties et auxiliaires du microcontrôleur; un double-clic sur un item ouvre la fenêtre *ad hoc*. La figure 7 illustre les items *Port x* où le détail des 8 bits du port (Px.0 en haut et Px.7 en bas) est donné à l'aide de DEL avec un code de position et de couleur : en haut et vert pour un bit à 1, en bas et rouge pour un bit à 0. Un clic droit sur une DEL ouvre un menu contextuel qui permet de positionner le bit associé :

- *No connection* pour un fil de port en sortie ;
- *Ground* pour un fil du port en entrée avec la valeur 0 ;
- *Vcc* pour un fil du port en entrée avec la valeur 1 (un double-clic sur une DEL permet d'alterner entre les états *Ground* et *Vcc*) ;
- *Net* pour relier un fil à un nœud (détails dans la suite du document).

Le champ *LATCH* correspond au registre Px qui permet de configurer une valeur hexadécimale de sortie du port correspondant (la mise à jour des données peut être obtenue à l'aide du bouton *Refresh* de la barre de tâche). Pour rappel, pour qu'un fil d'un port soit configuré en entrée, le bit correspondant doit être mis à 1.

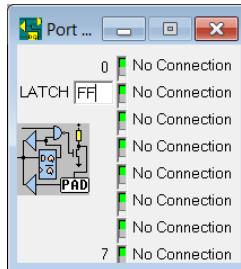


FIGURE 7 – Fenêtres de contrôle des entrées/sorties *Port x*

## 6.5. FENÊTRE PROGRAMME

En mode débogueur, la fenêtre Programme donne un certain nombre d'informations :

- toutes les instructions sont repérées par un point vert dans la marge (pour rappel, les directives d'assemblage ne sont pas codées en langage assembleur) ;
- si l'option *Display executed line* est validée, les instructions déjà exécutées voient leur point vert devenir bleu ;
- la ligne surlignée en bleu correspond à l'instruction qui va être exécutée (elle correspond à l'instruction pointée en mémoire programme par le PC) ;
- un point rouge avec un S, dans la marge, indique la position d'un point d'arrêt au niveau de l'instruction correspondante ;
- un T vert dans la marge indique la position d'une sonde.

## 6.6. FENÊTRE *Watches*

La fenêtre *Watches* permet de suivre l'évolution des registres, les bits des registres accessibles par bit et les symboles définis par DATA ou BIT, en positionnant des sondes d'observation :

1. Faire un clic droit au niveau de la fenêtre *Watches* ;
2. Sélectionner *Add* dans le menu contextuel ;
3. Renseigner le nom de la variable dans le champ *Expression* (il est possible d'utiliser le nom d'un registre, du bit ou du symbole, ou de la trouver à l'aide de la liste déroulante) ;
4. Cliquer sur le bouton *OK* et la variable apparaît dans la fenêtre *Watches*.

## 6.7. OBSERVATION TEMPORELLE

Il est possible de représenter l'évolution d'une variable (registre, bit d'un registre accessible par bit, ou symbole défini par DATA ou BIT) à l'aide d'un chronogramme (*Trace*) :

1. Afficher la fenêtre *Trace* à l'aide du menu *View* ▶ *Trace* ▶ *View*.
2. Configurer l'affichage par le menu *View* ▶ *Trace* ▶ *Options*. Une fenêtre apparaît alors (voir figure 8) où il est possible de configurer le mode de fonctionnement (arrêt, continu, sur changement, etc.), le nombre de cycles machines visualisés et si le tracé est de type déroulant (*Rolling trace*).



FIGURE 8 – Fenêtre de configuration de l'observation temporelle

3. Ajouter une sonde d'observation (voir section 6.6. Fenêtre *Watches*).
4. Visualiser une variable dans la fenêtre *Trace* :
  - (a) dans la fenêtre *Watches*, sélectionner une variable et faire un clic droit avec la souris ;
  - (b) dans le menu contextuel qui apparaît, sélectionner *Add/Delete from Trace List* (la même opération permet de supprimer une variable de la fenêtre *Trace*) ;
  - (c) au niveau du bandeau supérieur de la fenêtre *Trace*, sélectionner les variables à visualiser<sup>4</sup>.
4. L'appui sur le bouton *OK* de la fenêtre de configuration de l'affichage désélectionne toutes les variables.

## 6.8. GÉNÉRATEUR DE FONCTION

Ride permet de simuler des entrées qui évoluent dans le temps à l'aide de générateurs de fonctions. Leur utilisation passe par deux étapes, la première consiste en l'initialisation du générateur :

1. Ouvrir le menu *View ▶ Function Generators... ▶ Options* et cliquer sur le bouton *New* ;
2. Donner un nom au générateur dans le champ *Name* ;
3. Indiquer le type de la fonction : fonction (*Wave Form*) ou une expression (*Expression Sampling*).  
4. Cliquer sur le bouton *OK*.
5. Cliquer sur le bouton *CLOSE*.

Dans le cas d'un signal rectangulaire, il faut choisir le type *Wave Form* et remplir le champ *Expression to evaluate* de la façon suivante :

- indiquer la durée des états haut (H, 1 logique) et bas (L, 0 logique) en seconde (les sous-multiples milli et micro sont respectivement notés M et U) ;
- mettre l'expression entre parenthèses pour indiquer la périodicité de celle-ci ;
- par exemple, (L1M,H500U) permet de générer un signal rectangulaire (donc périodique) dont l'état bas dure 1 ms et l'état haut 500 µs.

Il est possible de modifier les paramètres d'un générateur à l'aide du menu *View ▶ Function Generators... ▶ Options*, puis en sélectionnant le générateur dans la liste et en cliquant sur le bouton *Edit*.

La deuxième étape consiste à relier le générateur à un fil d'un port d'entrées/sorties à l'aide d'un nœud (*net*) :

1. Ouvrir la fenêtre *Nets* à l'aide du menu *View ▶ Nets* ;
2. Créer un nouveau nœud en cliquant sur le bouton *New*. Il est possible de le renommer en cliquant sur le bouton  *Rename* ;
3. Associer le générateur à une ou des broches. Pour cela, il faut sélectionner chaque élément souhaité de la liste *Available* et le mettre dans la liste *Connected* à l'aide du bouton *>*. Un item de la liste *Connected* peut être enlevé en le sélectionnant et en cliquant sur le bouton *<*.

# ANNEXES

## A.1. REPRÉSENTATION DES NOMBRES

En informatique, il est courant de représenter les nombres dans trois bases différentes, la base indiquant le nombre de symboles (digit) disponibles dans la représentation :

- le décimal ou base 10 ;
- le binaire ou base 2 ;
- l'hexadécimal ou base 16.

Quelque soit la base utilisée, la représentation d'un nombre repose sur le même principe :

$$N = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b^1 + a_0 b^0 \quad (\text{A.1})$$

où  $b_n$  représente le poids du rang  $n$  dans la base  $b$  et  $a_n$  le digit du rang  $n$ . En effet, 2019 en décimal peut se décomposer ainsi :  $2 \times 10^4 + 0 \times 10^3 + 1 \times 10^1 + 9 \times 10^0$ .

De plus, l'incrémentation d'un rang alors que la valeur du digit est maximum (symbole le plus élevé) s'effectue en repassant ce digit à 0 et en incrémentant le digit de rang supérieur : c'est le principe de la retenue. En effet  $9 + 1 = 09 + 1 = 10$  où le rang 0 est à son maximum, il passe donc à 0 et le rang supérieur (rang 1) est incrémenté (ne pas oublié qu'il est possible d'ajouter autant de zéros que souhaité avant tout nombre).

### A.1.1. REPRÉSENTATION BINAIRE

En binaire, il n'y a que deux symboles : 0 et 1 et chaque digit est appelé bit. C'est la base naturelle des microporcesseurs (langage machine) puisqu'à l'origine, ils étaient constitué de relais qui avait deux états possibles : ouvert ou fermé. De nos jours, les transistors ont pris la place de ces relais, mais le principe de fonctionnement en tout ou rien demeure.

Si on se réfère à ce qui a été mentionné précédemment, les premiers nombres en binaire sont : 0, 1, 10 (le rang 0 étant précédemment au maximum, on incrémenté le rang 1), 11, 100, etc.

Le codage d'un nombre décimal ( $N$ )<sub>10</sub> en binaire ( $N$ )<sub>2</sub> peut se faire ainsi :

1. Effectuer la division euclidienne  $(N)_{10} / 2^n$ , en commençant par le rang le plus élevé possible (donnant une partie entière non nulle).
2. La partie entière de la division correspond au poids du rang  $n$ .
3. Effectuer une division euclidienne entre le reste de la division précédente et le rang inférieur  $(\text{Reste})_{10} / 2^{n-1}$ .
4. Recommencer à l'étape 2 jusqu'au rang 0.

Par exemple :

1.  $(77)_{10} / 2^7 \approx 0,6$  et  $(77)_{10} / 2^6 \approx 1,2$ .
2. Les poids respectifs des rangs 7 et 6 sont 0 et 1.
3. Le reste de  $(77)_{10} / 2^6$  vaut  $(13)_{10}$ , il faut donc effectuer le calcul  $(13)_{10} / 2^5$ .
4. Ainsi de suite. Le tableau A.1 donne le résultat pour chaque rang, la partie entière de chaque division ne pouvant valoir que 0 ou 1 en binaire.

<b>Rang</b>	7	6	5	4	3	2	1	0
<b>Poids (<math>2^n</math>)</b>	128	64	32	16	8	4	2	1
<b>Bit</b>	0	1	0	0	1	1	0	1

TABLEAU A.1 – Codage en base 2 de  $(77)_{10}$

On trouve ainsi que  $(77)_{10} = (1001101)_2$ .

Le bit de rang 0 est appelé bit de poids faible (LSB pour *Least Significant Bit*) et le bit de rang le plus élevé (dans un format donné) bit de poids fort (MSB pour *Most Significant Bit*). Pour  $(77)_{10}$ , avec un format 8 bits, le MSB vaut 0 (rang 7) et le LSB vaut 1 (rang 0).

Le décodage d'un nombre binaire en décimal se fait simplement par application de l'équation A.1. Par exemple,  $(1011)_2 = 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 = 8 + 2 + 1 = (11)_{10}$ .

### A.1.2. REPRÉSENTATION HEXADÉCIMALE

En hexadécimal, il n'y a 16 symboles : 0 à 9 et A à F (A correspondant à 10 en décimal et F à 15). Il permet une représentation plus pratique du binaire puisque chaque symbole en hexadécimal regroupe 4 bits ( $2^4 = 16$ ). Ainsi, 1 octet s'écrit avec 8 bits mais seulement 2 digits en hexadécimal.

Le passage d'un nombre décimal en hexadécimal peut se faire par la même méthode de codage qu'en binaire, mais en remplaçant  $2^n$  par  $16^n$  (cette méthode est généralisable à toute base  $b$  en effectuant des divisions par  $b^n$ ). Pour  $(77)_{10}$ , on trouve :

1.  $(77)_{10}/16^2 \approx 0,3$  et  $(77)_{10}/16^1 \approx 4,8$ .
2. Les poids respectifs des rangs 2 et 1 sont 0 et 4.
3. Le reste de  $(77)_{10}/16^1$  vaut  $(13)_{10}$ , il faut donc effectuer le calcul  $(13)_{10}/16^0$ .
4. La division par 1 (rang 0) est superflue, le digit de ce rang est donc  $(13)_{10} = (D)_{16}$ .

On trouve ainsi que  $(77)_{10} = (4D)_{16}$ .

Il est aussi possible de coder à partir de la représentation binaire en regroupant les bits par quatre et en leur attribuant un poids de  $2^n$  avec  $n$  allant de 0 à 3. Le tableau A.2 donne le détail de ces opérations pour  $(77)_{10} = 1001101_2$ , on retrouve  $(77)_{10} = (4D)_{16}$

<b>Base 2</b>	0	1	0	0	1	1	0	1
<b>rang</b>	3	2	1	0	3	2	1	0
<b>Poids <math>2^n</math></b>	8	4	2	1	8	4	2	1
<b>Base 16</b>				4			D	

TABLEAU A.2 – Codage en base 16 de  $77_{10}$

Le décodage d'un nombre hexadécimal en décimal se fait simplement par application de l'équation A.1. Par exemple,  $(4EB)_2 = 4 \times 16^2 + 14 \times 16^1 + 11 \times 16^0 = (1259)_{10}$ .

### A.1.3. REPRÉSENTATION DES NOMBRES SIGNÉS

En binaire, les nombres signés sont représentés en affectant le bit de poids fort au signe : un 0 indique un nombre positif et un 1, un nombre négatif. Le passage d'une valeur positive à sa valeur négative, et inversement, s'effectue à l'aide du complément à 2 (aussi appelé complément vrai, C2). Cette opération passe par trois étapes :

1. Représenter le nombre dans le format désiré (en tenant compte du bit de signe).
2. Calculer le complément à 1 (ou complément restreint, C1) en inversant tous les bits.
3. Ajouter 1 au résultat obtenu.

Par exemple, le codage de  $(-44)_{10}$  en binaire sous un format octet donne :

1.  $(44)_{10} = (00101100)_2$ , sur un format 8 bit (le MSB à 0 indique un nombre positif).
2.  $C1 = 11010011$
3.  $C2 = C1 + 1 = 11010100$  (le MSB à 1 indique bien un nombre négatif).

On trouve donc que  $(-44)_{10} = (11010100)_2$ . En faisant les mêmes opérations avec ce résultat, on trouverai la valeur absolue de ce nombre négatif.

En hexadécimal, il est possible d'utiliser le même principe en calculant le complément à 16 mais il est plus facile de coder le chiffre à partir de sa valeur binaire. Avec l'exemple précédent, on détermine aisément que  $(-44)_{10} = (11010100)_2 = (D4)_{16}$ .

Le tableau A.3 donne les correspondances entre binaire, hexadécimal et décimal signé ou non des seize premières valeurs.

Base 10	Base 2	Base 16	Signé
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	-8
9	1001	9	-7
10	1010	A	-6
11	1011	B	-5
12	1100	C	-4
13	1101	D	-3
14	1110	E	-2
15	1111	F	-1

TABLEAU A.3 – Nombres de 0 à 15 dans différentes représentations

### A.1.4. CODAGE DCB

Le décimal codé binaire (DCB ou BCD pour *Binary Coded Decimal*) est une représentation des nombres en binaire qui se rapproche de la représentation décimale. En effet, chaque digit décimal est codé sur 4 bits. Par exemple,  $(325)_{10}$  est codé en DCB vaut  $(001\ 100\ 101\ 001)_2$ .

La représentation hexadécimal du DCB est telle que  $(N)_{16} = (N)_{10}$ , soit, par exemple,  $(325)_{10} = (325)_{16}$ .

## A.2. MULTIPLE EN BINAIRE

Le Bureau international des poids et mesures (BIPM) est l'organisation intergouvernementale créée par la Convention du Mètre ; ses États Membres agissent en commun en ce qui concerne les sujets liés à la science des mesures et aux étalons de mesure. Le Système international d'unités (reconnu au niveau international sous l'abréviation de SI) est le système pratique d'unités de mesure recommandées. Le SI est défini et présenté dans la Brochure sur le SI, publiée par le BIPM [7]. Concernant les multiples et sous-multiples, cette brochure mentionne :

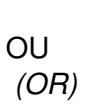
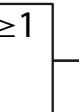
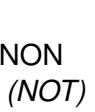
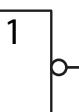
« Les préfixes SI représentent strictement des puissances de 10. Ils ne doivent pas être utilisés pour exprimer des puissances de 2 (par exemple, un kilobit représente 1000 bits et non 1024 bits).

Les préfixes adoptés par la CEI pour les puissances binaires sont publiés dans la norme internationale CEI 60027-2 : 2005, 3<sup>e</sup> édition, *Symboles littéraux à utiliser en électrotechnique — Partie 2 : Télécommunications et électronique*. Les noms et symboles des préfixes correspondant à  $2^{10}$ ,  $2^{20}$ ,  $2^{30}$ ,  $2^{40}$ ,  $2^{50}$  et  $2^{60}$  sont, respectivement : kibi, Ki ; mébi, Mi ; gibi, Gi ; tébi, Ti ; pébi, Pi ; et exbi, Ei. Ainsi, par exemple, un kibioctet s'écrit : 1 KiB =  $2^{10}$  B = 1024 B, où B désigne l'octet.

Bien que ces préfixes n'appartiennent pas au SI, ils doivent être utilisés en informatique afin d'éviter un usage incorrect des préfixes SI. »

## A.3. FONCTIONS LOGIQUES DE BASE

En logique combinatoire, il existe sept fonctions de base qui sont détaillées dans le tableau A.4.

Fonction	Symboles EU	Symbole USA	Opération	TdV															
ET (AND)			$S = a \cdot b$	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>a</th><th>b</th><th>S</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	a	b	S	0	0	0	0	1	0	1	0	0	1	1	1
a	b	S																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
OU (OR)			$S = a + b$	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>a</th><th>b</th><th>S</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	a	b	S	0	0	0	0	1	1	1	0	1	1	1	1
a	b	S																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
NON (NOT)			$S = \bar{a}$	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>a</th><th>S</th></tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>	a	S	0	1	1	0									
a	S																		
0	1																		
1	0																		

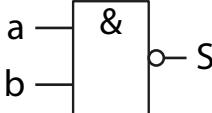
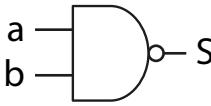
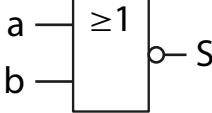
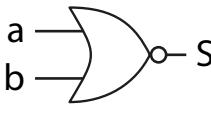
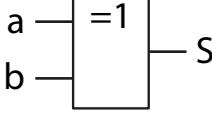
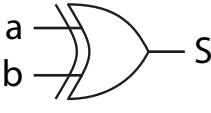
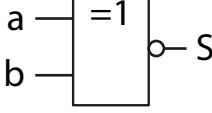
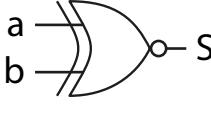
Fonction	Symboles EU	Symbole USA	Opération	TdV															
NON ET <i>NAND</i>			$S = \overline{a \cdot b}$	<table border="1"> <thead> <tr> <th>a</th><th>b</th><th>S</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	a	b	S	0	0	1	0	1	1	1	0	1	1	1	0
a	b	S																	
0	0	1																	
0	1	1																	
1	0	1																	
1	1	0																	
NON OU <i>(NOR)</i>			$S = \overline{a + b}$	<table border="1"> <thead> <tr> <th>a</th><th>b</th><th>S</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	a	b	S	0	0	1	0	1	0	1	0	0	1	1	0
a	b	S																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	0																	
OU EXCLUSIF <i>(XOR)</i>			$S = a \oplus b$	<table border="1"> <thead> <tr> <th>a</th><th>b</th><th>S</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	a	b	S	0	0	0	0	1	1	1	0	1	1	1	0
a	b	S																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	
NON OU EXCLUSIF <i>(XNOR)</i>			$S = a \odot b$	<table border="1"> <thead> <tr> <th>a</th><th>b</th><th>S</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	a	b	S	0	0	1	0	1	0	1	0	0	1	1	1
a	b	S																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	1																	

TABLEAU A.4 – Fonctions logiques de base

## A.4. CODE ASCII

L'*American Standard Code for Information Interchange* (ASCII) est une norme informatique de codage de caractères apparue dans les années 1960. L'ASCII définit 128 codes à 7 bits, comprenant 33 commandes de contrôle de terminal informatique (caractère NUL, tabulation, etc.) et 95 caractères imprimables : les chiffres arabes de 0 à 9, les lettres minuscules et capitales de A à Z, et des symboles mathématiques, de ponctuation et autre<sup>1</sup> (voir table A.1)

1. Source : wikipédia

Decimal Hex Char	Decimal Hex Char	Decimal Hex Char	Decimal Hex Char	Decimal Hex Char	Decimal Hex Char	Decimal Hex Char	Decimal Hex Char
0 0 [NULL]	32 20 [SPACE]	64 40 @	96 60 `				
1 1 [START OF HEADING]	33 21 !	65 41 A	97 61 a				
2 2 [START OF TEXT]	34 22 "	66 42 B	98 62 b				
3 3 [END OF TEXT]	35 23 #	67 43 C	99 63 c				
4 4 [END OF TRANSMISSION]	36 24 \$	68 44 D	100 64 d				
5 5 [ENQUIRY]	37 25 %	69 45 E	101 65 e				
6 6 [ACKNOWLEDGE]	38 26 &	70 46 F	102 66 f				
7 7 [BELL]	39 27 -	71 47 G	103 67 g				
8 8 [BACKSPACE]	40 28 (	72 48 H	104 68 h				
9 9 [HORIZONTAL TAB]	41 29 )	73 49 I	105 69 i				
10 A [LINE FEED]	42 2A *	74 4A J	106 6A j				
11 B [VERTICAL TAB]	43 2B +	75 4B K	107 6B k				
12 C [FORM FEED]	44 2C ,	76 4C L	108 6C l				
13 D [CARRIAGE RETURN]	45 2D -	77 4D M	109 6D m				
14 E [SHIFT OUT]	46 2E .	78 4E N	110 6E n				
15 F [SHIFT IN]	47 2F /	79 4F O	111 6F o				
16 10 [DATA LINK ESCAPE]	48 30 0	80 50 P	112 70 p				
17 11 [DEVICE CONTROL 1]	49 31 1	81 51 Q	113 71 q				
18 12 [DEVICE CONTROL 2]	50 32 2	82 52 R	114 72 r				
19 13 [DEVICE CONTROL 3]	51 33 3	83 53 S	115 73 s				
20 14 [DEVICE CONTROL 4]	52 34 4	84 54 T	116 74 t				
21 15 [NEGATIVE ACKNOWLEDGE]	53 35 5	85 55 U	117 75 u				
22 16 [SYNCHRONOUS IDLE]	54 36 6	86 56 V	118 76 v				
23 17 [END OF TRANS. BLOCK]	55 37 7	87 57 W	119 77 w				
24 18 [CANCEL]	56 38 8	88 58 X	120 78 x				
25 19 [END OF MEDIUM]	57 39 9	89 59 Y	121 79 y				
26 1A [SUBSTITUTE]	58 3A ..	90 5A Z	122 7A z				
27 1B [ESCAPE]	59 3B ,	91 5B _	123 7B {				
28 1C [FILE SEPARATOR]	60 3C V	92 5C }	124 7C }				
29 1D [GROUP SEPARATOR]	61 3D =	93 5D 1	125 7D }				
30 1E [RECORD SEPARATOR]	62 3E >	94 5E ~	126 7E ~				
31 1F [UNIT SEPARATOR]	63 3F ?	95 5F -	127 7F -				

FIGURE A.1 – Codes ASCII

## BIBLIOGRAPHIE

- [1] Atmel 8051 Microcontrollers — Hardware Manual  
<http://ww1.microchip.com/downloads/en/DeviceDoc/doc4316.pdf>
- [2] AT89C51 — Datasheet  
<http://ww1.microchip.com/downloads/en/DeviceDoc/doc0265.pdf>
- [3] Atmel Flash Microcontroller — Memory Organization
- [4] AT89 Series — Hardware Description
- [5] Atmel — Instruction Set
- [6] MA-51 — Reference Manual. Raisonnance
- [7] Brochure sur le SI — Le Système international d'unités  
<https://www.bipm.org>





9, rue Charles Fourier  
91011 Evry Cedex  
France  
[www.telecom-sudparis.eu](http://www.telecom-sudparis.eu)

