5) Le microcontrôleur AT90s8535

AVERTISSEMENT ! Ce chapitre ne présente que quelques ressources du microcontrôleur. Pour une information plus complète se référer au document constructeur : *http://www.atmel.com/dyn/products/datasheets.asp?family_id=607*

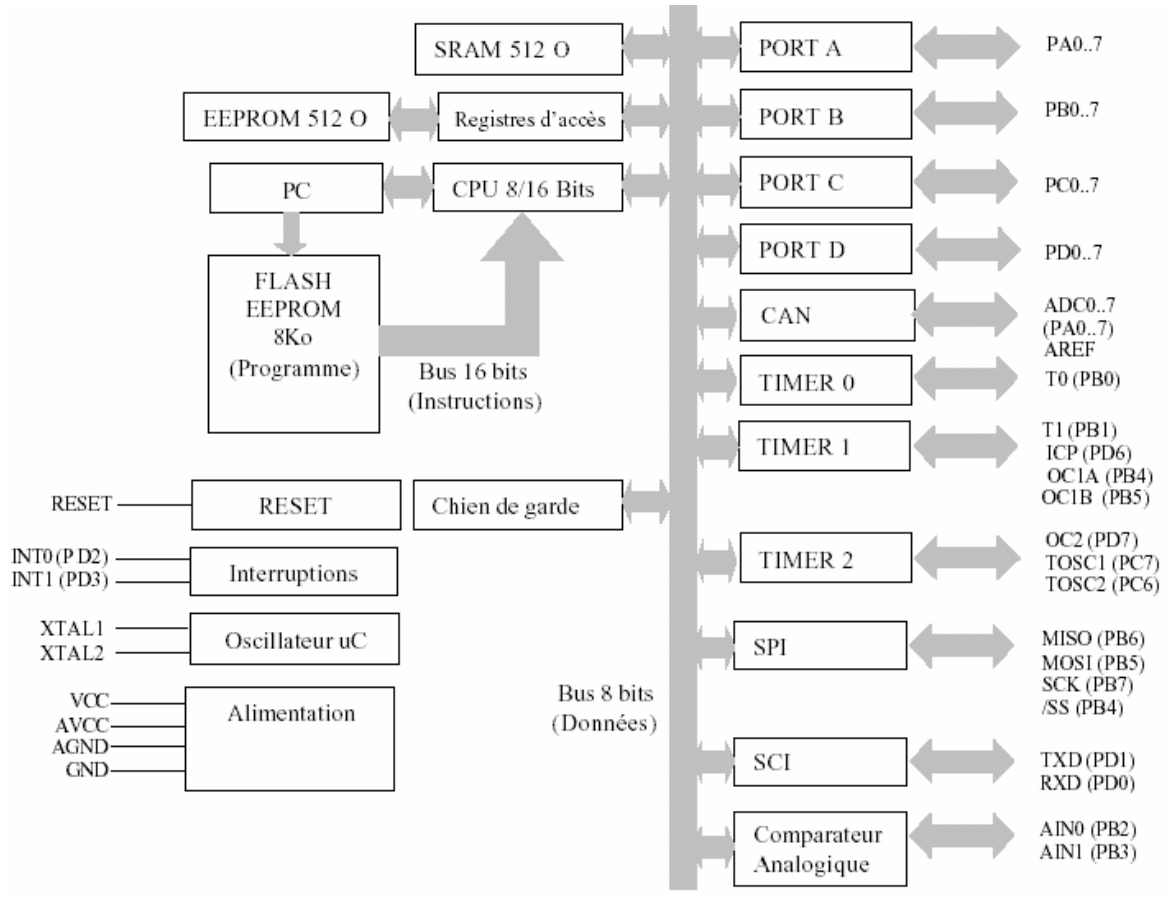
5.1) Présentation

Le circuit AT90S8535 est un microcontrôleur **8 bits** à architecture **RISC** produit par la société ATMEL. Il intègre un certain nombre de ressources et de périphériques lui permettant de couvrir un large éventail d'applications.

Caractéristiques principales :

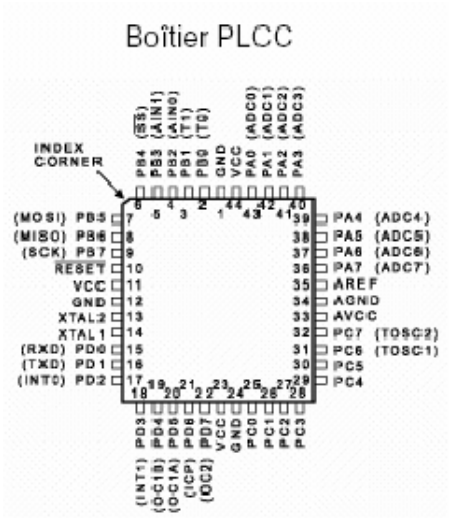
- CPU 8 bits capable d'exécuter la plupart des instructions en 1 cycle d'horloge
- Mémoire EEPROM de type flash de 8koctets programmable in situ (pour le programme applicatif)
- EEPROM de 512 octets à usage général
- RAM statique de 512 octets (stockage des variables, pile…)
- Un convertisseur analogique Numérique 10 bits à 8 entrées multiplexées
- Un comparateur de tensions analogiques
- Liaisons séries synchrone (SPI) et asynchrone (SCI)
- 2 timers 8 bits (dont un utilisable en horloge temps réel moyennant un oscillateur externe)
- 1 timer 16 bits
- 4 ports d'entrées/sorties parallèles 8 bits
- 2 entrées d'interruptions externes et une entrée de reset

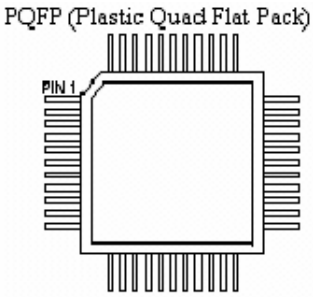
Architecture interne :

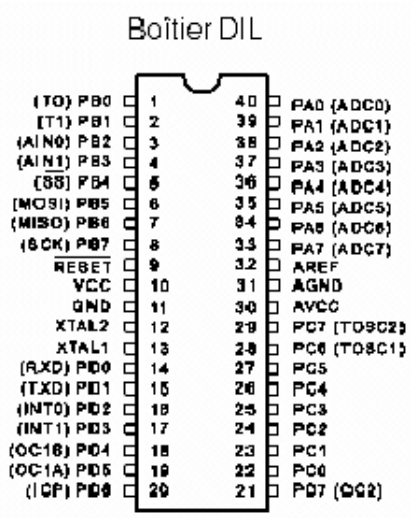


Encapsulation :

Le microcontrôleur est disponible dans plusieurs versions de boîtiers. Le boîtier de type MLF non représenté ici présente un encombrement de 7mm x 7mm.





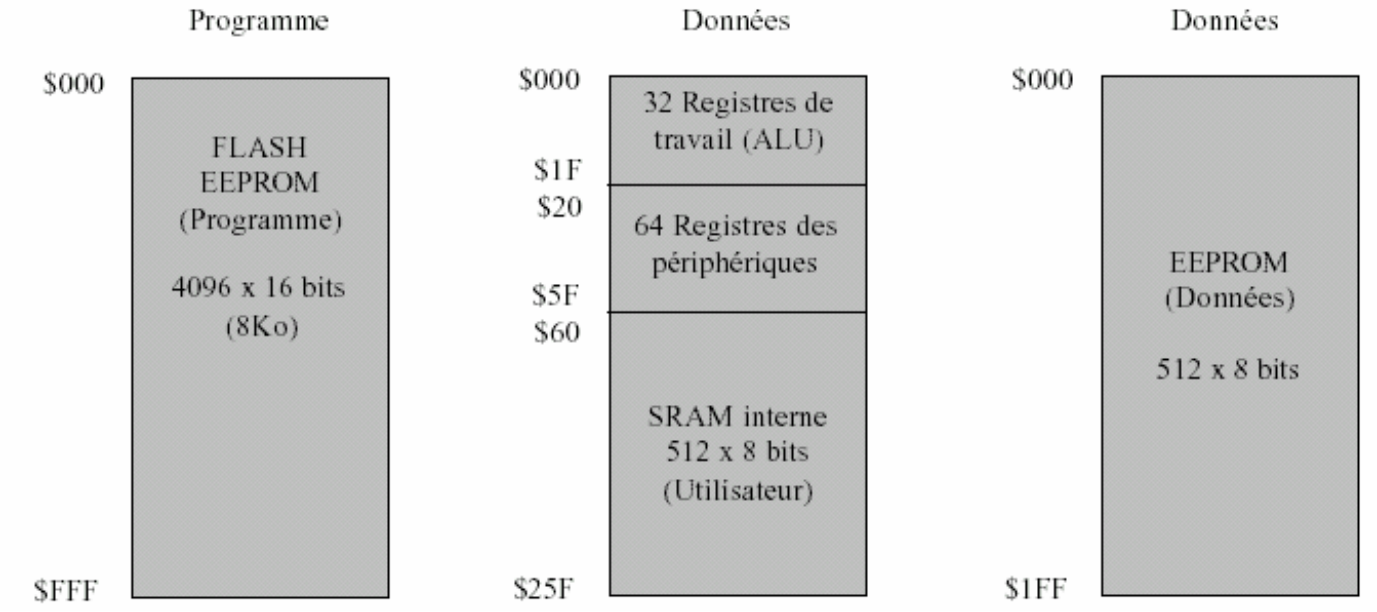


Mise en œuvre :

Dans sa configuration minimale le microcontrôleur ne nécessite qu'une alimentation, un circuit de « reset » et une horloge (horloge externe ou oscillateur à quartz). La plage d'alimentation va de 2.7V à 6V suivant les performances recherchées (économie d'énergie ou vitesse).

5.2) Organisation de la mémoire

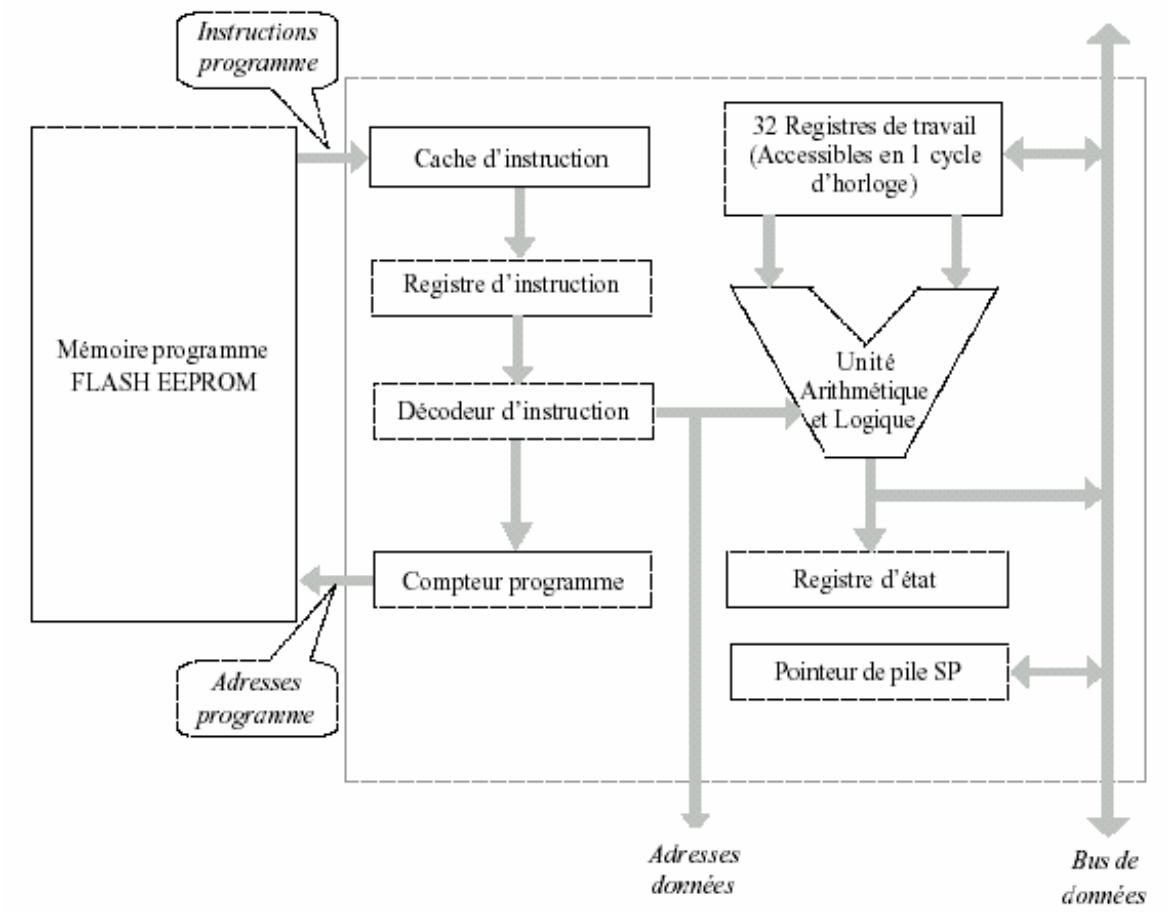
Les mémoires de données sont organisées en mots de 8 bits alors que la mémoire programme est organisée en mots de 16 bits. L'espace « données » est complétement séparé de l'espace « programme » (architecture de HARVARD), ce qui permet, grâce au fonctionnement en mode « pipe-line », une amélioration de la vitesse d'exécution des instructions.



5.3) Architecture de la partie « microprocesseur »

Celle-ci est constituée de :

- un ensemble de 32 registres de travail de 8 bits
- une unité arithmétique et logique (ALU)
- un compteur programme de 12 bits (soit 4K mots adressables)
- un registre d'état
- un pointeur de pile
- un cache, un registre et un décodeur d'instruction



5.3.1) Les registres de travail

Ils sont au nombre de 32 (notés R0 à R31) et peuvent s'apparenter à des registres contenant chacun 8 bascules D et possédant une commande de sortie 3 états (haute impédance) . (Voir chapitre 1.3.2.)

Attention !! Les instructions de manipulation de registres n'accèdent pas systématiquement aux registres R0 à R15. On préfèrera dans la mesure du possible travailler avec les registres R16 à R31.

Les registres :

- R26 et R27 concaténés constituent le registre d'index X
- R28 et R29 concaténés constituent le registre d'index Y
- R30 et R31 concaténés constituent le registre d'index Z

Les registres R26, R28, R30 constituent les poids faibles des index X, Y, Z.

5.3.2) L'unité arithmétique et logique

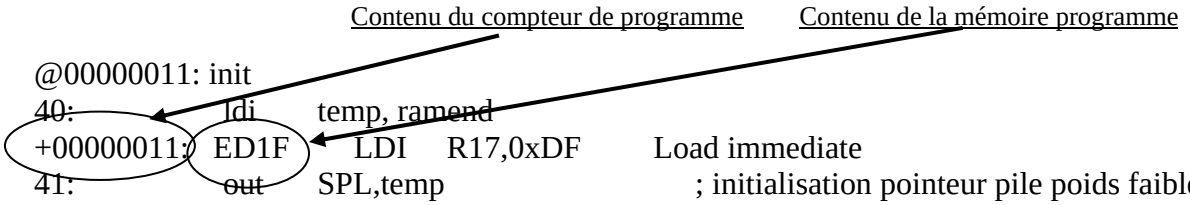
Elle effectue les opérations logiques et arithmétiques de base. (Voir chapitre 1.3.4)

5.3.3) Le compteur de programme

C’est un compteur de 12 bits à chargement parallèle (prépositionnable). Il adresse la mémoire de programme organisée en 4096 mots de 16 bits (les instructions étant codées sur 16 bits). A chaque top d’une horloge interne (dont la fréquence est proportionnelle à celle du Quartz) le compteur s’incrémente et vient pointer, s’il n’y a pas eu d’instruction de saut, l’adresse suivante de la mémoire de programme permettant ainsi le chargement et l’exécution de l’instruction qui s’y trouve. Les sauts de programme (ou sauts d’adresses) qui sont à l’initiative de l’instruction qui vient d’être exécutée sont rendus possibles grâce au chargement parallèle du compteur.

Exemple de programme et le contenu de la mémoire programme correspondant :

```
init:
        ldi     temp, ramend
        out     SPL,temp            ; initialisation pointeur pile poids faibles
        ser     temp
        out     DDRB,temp           ; port B en sortie
        out     portb, temp         ; reset leds
        ldi     temp,$60
```




```
@00000011: init
40:             ldi     temp, ramend
+00000011:  ED1F    LDI    R17,0xDF        Load immediate
41:             out     SPL,temp                ; initialisation pointeur pile poids faibles
+00000012:  BF1D     OUT    0x3D,R17        Out to I/O location
42:             ser     temp
+00000013:  EF1F     SER    R17             Load immediate
43:             out     DDRB,temp               ; port B en sortie
+00000014:  BB17     OUT    0x17,R17        Out to I/O location
44:             out portb, temp                 ; reset leds
+00000015:  BB18     OUT    0x18,R17        Out to I/O location
45:             ldi temp,$60
+00000016:  E610     LDI    R17,0x60        Load immediate
```

5.3.4) Le registre d’état SREG (Status Register)

Il s’agit d’un registre de 8 bits constitué par des bascules D qui peuvent être écrites ou lues séparément. Ce registre contient des « drapeaux » qui se positionnent ou non en fonction du résultat de l’instruction qui vient d’être exécutée (ils sont actifs à 1). La rubrique « Flags » du tableau des instructions (paragraphe 5.5) indique quels sont les drapeaux affectés en fonction des instructions exécutées. (Voir document annexe page 5/10 sur le registre SREG).

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
I	T	H	S	V	N	Z	C

I : si à 1 => autorise les interruptions
T : Bit de stokage pour la copie d'un bit
H : Drapeau de signalisation de semi-retenue
S : S = N ⊕ V (Drapeau de signalisation de signe arithmétique)
V : Drapeau de signalisation de double complément de dépassement
N : à 1 si l’instruction exécutée génère un résultat négatif
Z : à 1 si l’instruction exécutée génère un résultat nul

C : à 1 si l'instruction exécutée a provoqué une retenue
Remarque : Lorsqu'une instruction est susceptible de modifier l'état d'un bit du registre SREG, celui-ci est remis à 0 avant l'exécution de l'instruction.

5.3.5) Le pointeur de pile SP (Stack Pointer)

C'est un registre de 16 bits (SPH et SPL) situé aux adresses $5E et $5D et dont seulement les 10 bits de poids faibles sont utilisés (puisque la zone RAM va de $60 à $25F). Il pointe la prochaine adresse libre dans la pile (SRAM interne dont l'adresse finale est $25F). Son rôle est:

- d'adresser la pile pour la **sauvegarde** de l'adresse de retour du programme lors d'un appel à un sous-programme ou lors d'une interruption. (instruction CALL ou ICALL par exemple).
- d'adresser la pile pour la **restitution** de l'adresse de retour du programme lors du retour d' un sous-programme ou du retour d'une interruption. (instruction RET ou RETI).

Le pointeur de pile doit toujours être initialisé avec l'adresse haute de la RAM dès le début du programme. Dans le cas du AT90S8535 cette adresse est $25F.

Exemple d'évolution du pointeur de pile en fonction de l'évolution du programme:

	Pointeur de programme	Instruction associée	Valeur SP du pointeur de pile après exécution de l'instruction	Contenu de la pile
	$0011	Ldi temp,$5F	xxxx	($025F) = $xxxx
	$0012	Out SPL,temp	$xx5F	
	$0013	ldi temp,$02	$xx5F	
	$0014	Out SPH,temp	$025F	
	$0015	ser temp	$025F	
	$0016	out ddrc, temp	$025F	
	$0017	out portc,temp	$025F	
1 →	$0018	rcall init_timer1	$025D	($025F) = $0019
	$0019	rjmp debut		
	....	.....		
	$0029	init_timer1: ldi temp, $00	$025D	
	$002A	out tccr1a, temp	$025D	
	$002B	out ocr1ah, temp	$025D	
	$002C	ldi temp,78	$025D	
	$002D	out ocr1al, temp	$025D	
2 →	$002E	Ret	$025F	

$xx = octet non déterminé
($025F) = contenu de la mémoire à l'adresse $025F.

1 : lors de l'exécution de l'instruction RCALL, l'adresse de retour du programme ($0019) est stockée dans la pile à l'adresse indiquée par le pointeur ($025F) puis celui-ci est décrémenté de 2 ($025D).

2 : lors de l'exécution de l'instruction RET, le pointeur de pile est incrémenté de 2 ($025E, $025F) et le compteur de programme prend la valeur du contenu de la pile ($0019). La prochaine instruction exécutée sera par conséquent « rjmp debut ».

Remarque : le pointeur est décrémenté ou incrémenté deux fois à chaque appel ou retour de sous-programme parce que la mémoire RAM est organisée en mots de 8 bits.

5.4) Reset, interruptions et modes de veille

5.4.1 Le Reset

Il a pour but la réinitialisation complète du microcontrôleur. Celle-ci se produit :

- à la mise sous tension du circuit tant que les alimentations ne sont pas stabilisées
- à partir d'un circuit « chien de garde »
- ou par action d'un opérateur (bouton poussoir).

Souvent la génération du **Reset** à la mise sous tension est confiée à des circuits intégrés spécialisés (ex : Motorola MC34064) et parfois à un simple réseau RC. Par construction (câblage interne) **le passage à 0 de la broche Reset force le compteur de programme à pointer l'adresse 0.** Le signal de Reset doit être maintenu actif pendant une durée T après la mise sous tension (cette durée est fournie par le constructeur du composant et est de 50ns minimum s'agissant du AT90S8535).

5.4.2) Les interruptions

Il existe deux types d'interruptions :

- les interruptions externes (broches INT0 et INT1 du circuit)
- les interruptions internes provoquées par les périphériques intégrés.

Le rôle des interruptions est de provoquer une suspension de l'exécution du programme en cours pour traiter un programme devenu de priorité supérieure.
Les interruptions peuvent être masquées par le programmeur.
Comme pour le **Reset**, l'apparition d'une interruption force le compteur de programme à une adresse déterminée fixée par le câblage interne du composant.

Le tableau suivant recense les différentes adresses d'interruptions du circuit AT90S8535 :

Vecteur N°	Adresse	Source	Description
1	$0000	RESET	Reset par mise sous tension ou niveau bas sur broche Reset ou Chien de garde
2	$0001	INT0	Front ou niveau actif sur la broche INT0 quand INT0 est validée
3	$0002	INT1	Front ou niveau actif sur la broche INT1 quand INT1 est validée
4	$0003	TIMER2 COMP	Comparaison réussie du timer2
5	$0004	TIMER2 OVF	Dépassement du compteur du timer2
6	$0005	TIMER1 CAPT	Front actif sur l'entrée de capture ICP
7	$0006	TIMER1 COMPA	Comparaison réussie A du timer1
8	$0007	TIMER1 COMPB	Comparaison réussie A du timer1
9	$0008	TIMER1 OVF	Dépassement du compteur du timer1
10	$0009	TIMER0 OVF	Dépassement du compteur du timer0
11	$000A	SPI STC	Transfert par la SPI terminé
12	$000B	UART RX	Réception d'un octet terminée par l'UART
13	$000C	UART UDRE	Registre de transmission de l'UART vide
14	$000D	UART TX	Transmission d'un octet terminée par l'UART
15	$000E	ADC	Conversion analogique numérique terminée
16	$000F	EE_RDY	EEPROM prête (Ecriture d'un octet terminée)
17	$0010	ANA_COMP	Basculement de la sortie du comparateur analogique

Mécanismes de traitement d'une interruption :
Lors d'une interruption le processeur exécute la séquence suivante :

- le bit I du registre d'état (SREG) est mis à 0 (inhibition des autres sources d'interruption)
- l'exécution du programme en cours est interrompue
- le contenu du PC est sauvegardé dans la pile
- le PC est chargé avec la valeur de l'adresse d'interruption

- l'instruction de saut à l'adresse de traitement de l'interruption est exécutée
- le programme de traitement d'IT est exécuté
- à la fin du programme, l'instruction **RETI** restaure le contenu du PC
- le bit I du registre d'état (SREG) est mis à 1 (validation des autres sources d'interruption).

Remarque : lors d'une interruption, **le processeur ne sauvegarde pas le contexte** (état des registres de travail et registre d'état SREG). Il est donc important de sauvegarder au moins le registre d'état en début du sous-programme d'interruption et de le restituer à la fin en utilisant les instructions **PUSH** et **POP**.

Exemple pour la sauvegarde de SREG :

```
EXT_INT0:
        in temp, sreg           ;sauvegarde de SREG
        push temp
        ….
        pop temp                ;récup SREG
        out sreg, temp
        reti                    ;retour du sous-prog d'interruption
```

Exemple : reprenons le programme du paragraphe 5.3.5) et supposons que lors de son exécution une interruption provoquée par un évènement extérieur apparaisse sur la broche **INT0** du composant alors que les interruptions sont autorisées.

INT0 →

Pointeur de programme	Instruction associée	Valeur SP du pointeur de pile après exécution de l'instruction	Contenu de la pile
$0011	Ldi temp,$5F	xxxx	$xxxx
$0012	Out SPL,temp	$xx5F	
$0013	ldi temp,$02	$xx5F	
$0014	Out SPH,temp	$025F	
$0015	ser temp	$025F	
$0016	out ddrc, temp	$025F	
$0017	out portc,temp	$025F	
$0018	rcall init_timer1	$025D	($025F) = $0019
$0019	rjmp debut		...
....	.....		
$0029	init_timer1: ldi temp, $00	$025D	...
$002A	out tccr1a, temp	$025D	
$002B	out ocr1ah, temp	$025B	($025D) = $002C
$002C	ldi temp,78	$025D	
$002D	out ocr1al, temp	$025D	
$002E	Ret	$025F	
.....	....		
$0001	Rjmp @gest_int0	$025B	($025D) = $002C
...			
@Gest_int0	Instruction 1	$025B	
....	.....	.....	
....	Instruction n	$025B	
.....	**Reti**	$025D	

Lors de l'apparition de **INT0**, le processeur :

- interdit toute nouvelle interruption
- termine l'exécution de l'instruction en cours (out ocr1ah, temp)

- stocke l'adresse de retour ($002C) dans la pile à l'adresse $025D et charge le compteur de programme avec la valeur du vecteur associé à INT0 ($0001)
- décrémente le pointeur de pile de 2 ($025B)
- exécute l'instruction de branchement au sous-programme associé à INT0 (gest_int0) puis les instructions qui suivent.

Lors de l'exécution de **Reti**, le processeur :

- incrémente de 2 le pointeur de pile ($025D)
- charge le compteur de programme avec la valeur pointée par le pointeur de pile ($002C)
- ré-autorise les interruptions

5.4.3) Les modes veille

Ce sont des modes qui permettent d'économiser de l'énergie (utilisés dans des applications portables).

5.5) Le jeu d'instructions

l'AT90S8535 dispose d'un jeu d'instructions réduit (RISC) codé en majorité sur 16 bits, ce qui explique l'organisation de la mémoire programme en mots de 16 bits. La plupart de ces instructions sont exécutées en un cycle d'horloge.

Mnemonic	Operands	Description	Operation	Flags	Cycles
ADD	Rd,Rr	Add without Carry	Rd = Rd + Rr	Z,C,N,V,H,S	1
ADC	**Rd,Rr**	**Add with Carry**	**Rd = Rd + Rr + C**	**Z,C,N,V,H,S**	**1**
SUB	Rd,Rr	Subtract without Carry	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	Rd,K8	Subtract Immediate	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	Rd,Rr	Subtract with Carry	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	Rd,K8	Subtract with Carry Immedtiate	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	Rd,Rr	Logical AND	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Logical AND with Immediate	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Logical OR	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Logical OR with Immediate	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Logical Exclusive OR	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	One's Complement	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Two's Complement	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Set Bit(s) in Register	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Clear Bit(s) in Register	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Increment Register	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Decrement Register	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Test for Zero or Negative	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Clear Register	Rd = 0	Z,C,N,V,S	1
SER	Rd	Set Register	Rd = $FF	None	1
ADIW	Rdl,K6	Add Immediate to Word	Rdh:Rdl = Rdh:Rdl + K6	Z,C,N,V,S	2
SBIW	Rdl,K6	Subtract Immediate from Word	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2

Exemple :

ADC r17, r16 => les contenus de r16, r17 et de la retenue C sont ajoutés et le résultat est placé dans r17.

La colonne **« flags** » renseigne sur le résultat de l'opération. Par exemple, les registres étant sur 8 bits, si le résultat est sur 9 bits le bit de carry C est positionné à 1.

La colonne « **cycle** » indique le nombre de périodes d'horloge nécessaires pour exécuter l'instruction. Si le processeur est équipé d'une horloge à 8 Mhz alors l'instruction sera exécutée en 125 ns.
L'extrait ci-dessous issu de la documentation technique du constructeur indique le code binaire associé à l'instruction ADC (ADd with Carry).

Description:

Adds two registers and the contents of the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr + C

	Syntax:	Operands:	Program Counter:
(i)	ADC Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0001	11rd	dddd	rrrr

Instructions de saut et de branchement:

P= registre de périphérique. Rd et Rr= registre de travail

Mnemonic	Operands	Description	Operation	Flags	Cycles
RJMP	k	Relative Jump	PC = PC + k +1	None	2
IJMP	None	Indirect Jump to (Z)	PC = Z	None	2
RCALL	k	Relative Call Subroutine	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	None	Indirect Call to (Z)	STACK = PC+1, PC = Z	None	3/4*
CALL	k	Call Subroutine	STACK = PC+2, PC = k	None	4/5*
RET	None	Subroutine Return	PC = STACK	None	4/5*
RETI	None	Interrupt Return	PC = STACK	I	4/5*
CPSE	Rd,Rr	Compare, Skip if equal	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd -Rr	Z,C,N,V,H,S	1
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,H,S	1
CPI	Rd,K8	Compare with Immediate	Rd - K	Z,C,N,V,H,S	1
SBRC	Rr,b	Skip if bit in register cleared	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Skip if bit in register set	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Skip if bit in I/O register cleared	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Skip if bit in I/O register set	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3
BRBC	s,k	Branch if Status flag cleared	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Branch if Status flag set	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Branch if equal	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Branch if lower	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Branch if minus	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Branch if plus	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1	None	1/2

BRLT	k	Branch if less than (signed)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag set	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	if(I==0) PC = PC + k + 1	None	1/2

Exemples :
RJMP : instruction de saut relatif.

Operation:
(i) PC ← PC + k + 1

	Syntax:	Operands:	Program Counter:	Stack
(i)	RJMP k	-2K ≤ k < 2K	PC ← PC + k + 1	Unchanged

16-bit Opcode:

1100	kkkk	kkkk	kkkk

CPSE (ComPare and Skip if Equal) : compare deux registres et saute l'instruction qui suit si égalité.

Instructions de transfert de données

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X	Load Indirect	Rd = (X)	None	2*
LD	Rd,X+	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Load Indirect	Rd = (Y)	None	2*
LD	Rd,Y+	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,Z+	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,-Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Load Indirect with displacement	Rd = (Z+q)	None	2*
STS	k,Rr	Store Direct	(k) = Rr	None	2*
ST	X,Rr	Store Indirect	(X) = Rr	None	2*
ST	X+,Rr	Store Indirect and Post-Increment	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Store Indirect and Pre-Decrement	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Store Indirect	(Y) = Rr	None	2*
ST	Y+,Rr	Store Indirect and Post-Increment	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Store Indirect and Pre-Decrement	Y=Y-1, (Y) = Rr	None	2

STD	Y+q,Rr	Store Indirect with displacement	(Y+q) = Rr	None	2
ST	Z,Rr	Store Indirect	(Z) = Rr	None	2
ST	Z+,Rr	Store Indirect and Post-Increment	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Store Indirect and Pre-Decrement	Z=Z-1, (Z) = Rr	None	2
STD	Z+q,Rr	Store Indirect with displacement	(Z+q) = Rr	None	2
LPM	None	Load Program Memory	R0 = (Z)	None	3

Exemples :
LDI (LoaD Immediate)
LDI r16, $25 => charge le registre de travail r16 avec la valeur $25

Attention LDI ne fonctionne pas avec les registres r0 à r15 !!!

LDS (Load Direct from Sram)
LDS r16, $80 => charge r16 avec le contenu de la sram situé à l'adresse $80.

STS (Store To Sram)
STS $80, r16 => sauve le contenu de r16 à l'adresse $80 de la sram.

Adressage indirect :
LD r16, X => charge r16 avec le contenu de la sram situé à l'adresse contenue dans X. Il faut donc au préalable initialiser X.

Instructions de manipulations de bits :

P= registre de périphérique. Rd et Rr= registre de travail

Mnemonic	Operands	Description	Operation	Flags	Cycles
LSL	Rd	Logical shift left	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S	1
LSR	Rd	Logical shift right	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Rotate left through carry	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S	1
ROR	Rd	Rotate right through carry	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Arithmetic shift right	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Swap nibbles	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Set flag	SREG(s) = 1	SREG(s)	1
BCLR	s	Clear flag	SREG(s) = 0	SREG(s)	1
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2
BST	Rr,b	Bit store from register to T	T = Rr(b)	T	1
BLD	Rd,b	Bit load from register to T	Rd(b) = T	None	1
SEC	None	Set carry flag	C =1	C	1
CLC	None	Clear carry flag	C = 0	C	1
SEN	None	Set negative flag	N = 1	N	1
CLN	None	Clear negative flag	N = 0	N	1
SEZ	None	Set zero flag	Z = 1	Z	1
CLZ	None	Clear zero flag	Z = 0	Z	1
SEI	None	Set interrupt flag	I = 1	I	1
CLI	None	Clear interrupt flag	I = 0	I	1
SES	None	Set signed flag	S = 1	S	1
CLN	None	Clear signed flag	S = 0	S	1
SEV	None	Set overflow flag	V = 1	V	1

CLV	None	Clear overflow flag	V = 0	V	1
SET	None	Set T-flag	T = 1	T	1
CLT	None	Clear T-flag	T = 0	T	1
SEH	None	Set half carry flag	H = 1	H	1
CLH	None	Clear half carry flag	H = 0	H	1

Instructions d'accès direct aux registres des périphériques :
P= registre de périphérique. Rd et Rr = registre de travail

IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1

Instructions d'accès à la pile :

PUSH	Rr	Push register on Stack	STACK = Rr	None	2
POP	Rd	Pop register from Stack	Rd = STACK	None	2

Instructions spéciales :

NOP	None	No operation	None	None	1
SLEEP	None	Sleep	Mode économie d'énergie	None	1
WDR	None	Watchdog Reset	RAZ chien de garde	None	1

Rd: Destination (and source) register in the register file
Rr: Source register in the register file
b: Constant (0-7), can be a constant expression
s: Constant (0-7), can be a constant expression
P: Constant (0-31/63), can be a constant expression
K6; Constant (0-63), can be a constant expression
K8: Constant (0-255), can be a constant expression
k: Constant, value range depending on instruction. Can be a constant expression
q: Constant (0-63), can be a constant expression
Rdl: R24, R26, R28, R30. For ADIW and SBIW instructions
X,Y,Z: Indirect address registers (X=R27:R26, Y=R29:R28, Z=R31:R30)

5.6) Les directives d'assemblage

Ce sont des instructions qui ne sont pas traduites en code machine exécutable mais réservées au programme d'assemblage du fichier source. Elles représentent un ensemble d'utilitaires facilitant l'écriture du programme d'application. Le tableau ci-dessous récapitule la liste des directives supportées par l'assembleur des microcontrôleurs ATMEL.

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data Segment

DW	Define Constant word(s)
ENDM, ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn Macro expansion in list file on
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

Directives les plus utilisées :

- **.INCLUDE** : permet de lire une source d'informations issue d'un autre fichier. Cette directive est beaucoup utilisée pour spécifier le type de microcontrôleur mis en œuvre .

Exemple : **.include** « 8535def.inc » => spécifie à l'assembleur d'utiliser le fichier nommé « 8535def.inc ». Ce fichier est un fichier de description des ressources internes du microcontrôleur AT90S8535. Il permet au programmeur d'utiliser les mnémoniques des périphériques par exemple plutôt que leur adresse physique.

- **.ORG :** permet de forcer l'adresse d'implantation du début d'un programme.

Ex : **.ORG $100** => implante le programme en mémoire à partir de l'adresse hexadécimale 100.

- **.DEF :** permet d'assigner un registre à une variable.

Ex : **.DEF temp = R16** => la variable **temp** correspond au registre R16.

- **.EQU :** permet d'assigner une valeur à une étiquette.

Ex : **.EQU ddrb = $17** => la variable ddrb prend la valeur hexadécimale 17.

- **.CSEG** : définit un segment qui sera implanté dans la mémoire Flash. A l'assemblage le texte qui suit sera traité comme une suite d'instructions.

Ex : **.CSEG**
```
.org 0
rjmp    RESET
.org 14
rjmp    RX_COMPLETE_INT
```

- **.DB** : définit un segment de données. L'assembleur ne les traitera pas comme des instructions.

Ex : .cseg
```
.org 0x100

sine:           ; 256 step sinewave table (one period)
.db             0x80,0x83,0x86,0x89,0x8c,0x8f,0x92,0x95,0x98,
```

L’exemple ci-dessus permet de ranger des données (un tableau) dans la mémoire Flash à partir de l’adresse 0x100 . Si on remplace la directive **.cseg** par la directive **.eseg** les données seront rangées dans la mémoire eeprom.

5.7) Les périphériques

5.7.1 Les registres associés aux périphériques

Adresse	Nom	Description
$00 ($20)		Réservé : Ne pas utiliser !
$01 ($21)		Réservé : Ne pas utiliser !
$02 ($22)		Réservé : Ne pas utiliser !
$03 ($23)		Réservé : Ne pas utiliser !
$04 ($24)	ADCL	Registre de données CAN (Pds faibles)
$05 ($25)	ADCH	Registre de données CAN (Pds forts)
$06 ($26)	ADCSR	Registre de contrôle du CAN
$07 ($27)	ADMUX	Registre de sélection entrée CAN
$08 ($28)	ACSR	Registre de ctrle du comparateur ana
$09 ($29)	UBRR	Registre de vitesse en baud de l'UART
$0A ($2A)	UCR	Registre de contrôle de l'UART
$0B ($2B)	USR	Registre de statut de l'UART
$0C ($2C)	UDR	Registre de données de l'UART
$0D ($2D)	SPCR	Registre de contrôle de la SPI
$0E ($2E)	SPSR	Registre de statut de la SPI
$0F ($2F)	SPDR	Registre de données de la SPI
$10 ($30)	PIND	Etat des broches d'entrée du PORTD
$11 ($31)	DDRD	Direction des broches du PORTD
$12 ($32)	PORTD	Etat des broches de sortie du PORTD
$13 ($33)	PINC	Etat des broche d'entrée du PORTC
$14 ($34)	DDRC	Direction des broches du PORTC
$15 ($35)	PORTC	Etat des broches de s ortie du PORTC
$16 ($36)	PINB	Etat des broches d'entrée du PORTB
$17 ($37)	DDRB	Direction des broches du PORTB
$18 ($38)	PORTB	Etat des broches de sortie du PORTB
$19 ($39)	PINA	Etat des broche d'entrée du PORTA
$1A ($3A)	DDRA	Direction des broches du PORTA
$1B ($3B)	PORTA	Etat des broches de sortie du PORTA
$1C ($3C)	EECR	Registre de contrôle de l' EEPROM
$1D ($3D)	EEDR	Registre de données de l'EEPROM
$1E($3E)	EEARL	Registre d'adresse EEPROM (Pds faibles)
$1F ($3F)	EEARH	Registre d'adresse EEPROM (Pds faibles)

Adresse	Nom	Description
$20 ($40)		Réservé : Ne pas utiliser !
$21 ($41)	WDCTR	Registre de contrôle du chien de garde
$22 ($42)	ASSR	Statut des accès Timer 2 en mode RTC
$23 ($43)	OCR2	Registre de comparaison du timer 2
$24 ($44)	TCNT2	Compteur du timer 2
$25 ($45)	TCCR2	Registre de contrôle du timer 2
$26 ($46)	ICR1L	Registre de capture du timer 1 (Pds faibles)
$27 ($47)	ICR1H	Registre de capture du timer 1 (Pds forts)
$28 ($48)	OCR1BL	Registre de comparaison B du timer 1 (LSB)
$29 ($49)	OCR1BH	Registre de comparaison B du timer 1 (MSB)
$2A ($4A)	OCR1AL	Registre de comparaison A du timer 1 (LSB)
$2B ($4B)	OCR1AH	Registre de comparaison A du timer 1 (MSB)
$2C ($4C)	TCNT1L	Compteur du timer 1 (Pds faibles)
$2D ($4D)	TCNT1H	Compteur du timer 1 (Pds forts)
$2E ($4E)	TCCR1B	Registre de contrôle B du timer 1
$2F ($4F)	TCCR1A	Registre de contrôle A du timer 1
$30 ($50)		Réservé : Ne pas utiliser !
$31 ($51)		Réservé : Ne pas utiliser !
$32 ($52)	TCNT0	Compteur du timer 0
$33 ($53)	TCCR0	Registre de contrôle du timer 0
$34 ($54)	MCUSR	Registre d'état général du processeur
$35 ($55)	MCUSR	Registre de contrôle général du processeur
$36 ($56)		Réservé : Ne pas utiliser !
$37 ($57)		Réservé : Ne pas utiliser !
$38 ($58)	TIFR	Registre d'état des interruptions TIMER
$39 ($59)	TIMSK	Registre de validation des interruptions TIMER
$3A ($5A)	GIFR	Registre d'état des interruptions externes
$3B ($5B)	GIMSK	Registre de validation des interruptions externes
$3C ($5C)		Réservé : Ne pas utiliser !
$3D ($5D)	SPL	Pointeur de pile (Pds faibles)
$3E($5E)	SPH	Pointeur de pile (Pds forts)
$3F ($5F)	SREG	Registre d'état de la CPU

Les adresses qui ne sont pas placées entre parenthèses sont les adresses à utiliser par les instructions d'accès direct aux registres des périphériques : **IN** et **OUT**.

5.7.2) Exemple de périphériques : les ports d'interfaces parallèles

L'AT90S8535 dispose de 4 ports d'interfaces de 8 bits chacun accessibles au travers de 3 registres de 8 bits. Cependant certaines broches de ces ports peuvent être partagées avec d'autres ressources internes. **C'est la phase d'initialisation qui permet de déterminer quelles seront les ressources mises en œuvre**.
Chaque port dispose de :

- un registre de direction **DDRx** qui spécifie si le port est utilisé en entrée ou en sortie (un 1 dans un de ses bits signifie que la broche correspondante est en sortie).
- Un registre de données entrantes **PINx** : l'état des bits de ce registre correspond aux niveaux logiques sur les broches programmées en entrée.
- Un registre de données sortantes ou d'options **PORTx** : en sortie il fixe le niveau logique sur la broche correspondante. Si la broche a été configurée en entrée, le port permet de valider ou non la résistance de pull-up interne.

Les registres PINx permettent au microcontrôleur de lire l'état logique des broches. Les registres PORTx permettent au microcontrôleur de forcer à un niveau logique les broches des ports (commande d'actionneurs « tout ou rien »).

Description des registres :

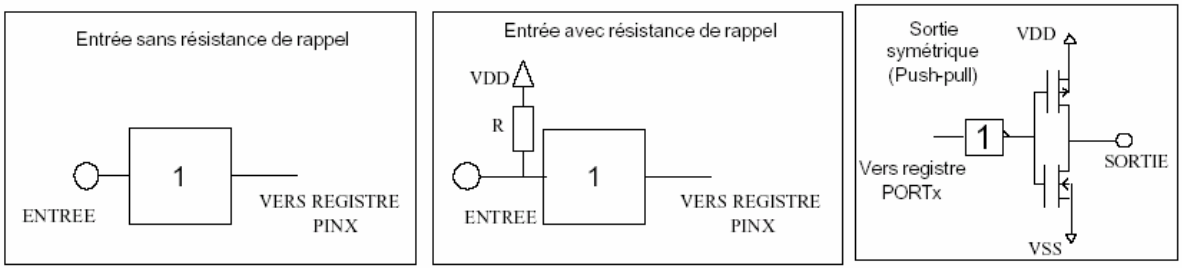
	Registre de Direction et adresse	Registre de données entrantes et adresse	Registre de données sortantes et adresse	Nbre de bits
Port A	DDRA $1A ($3A)	PINA $19 ($39)	PORTA $1B ($3B)	8
Port B	DDRB $17 ($37)	PINB $16 ($36)	PORTB $18 ($38)	8
Port C	DDRC $14 ($34)	PINC $13 ($33)	PORTC $15 ($35)	8
Port D	DDRD $11 ($31)	PIND $10 ($30)	PORTD $12 ($32)	8

Remarque : La 1ère adresse correspond à l'adresse à utiliser avec les instructions OUT et IN, la 2ème adresse donnée entre parenthèses correspond à l'adresse réelle du registre. Cette 2ème adresse est celle à utiliser lors des accès au registre par les instructions d'accès en RAM (STS par exemple).

Configuration des broches :

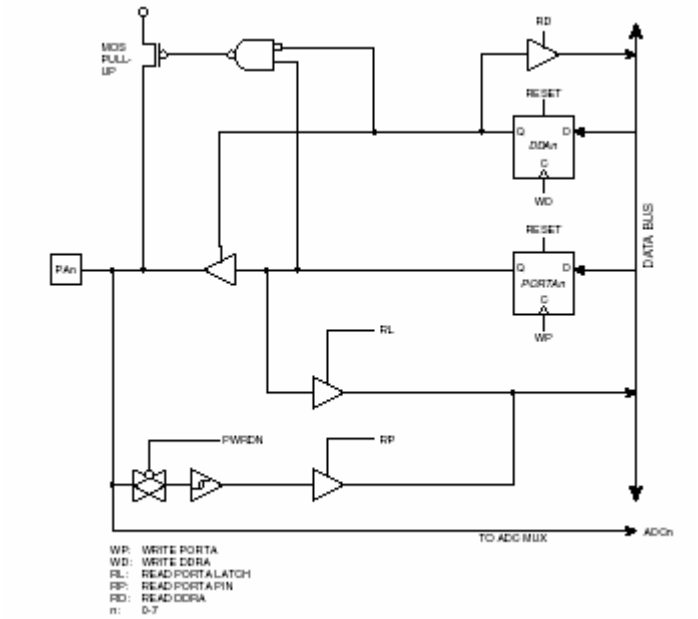
Bit du DDRx	Bit du PORTx	Bit du PINx	Direction	Configuration
0	0	X_{IN}	Entrée	Sans Pull-Up. (X_{IN} est l'image du niveau présent sur la broche)
0	1	X_{IN}	Entrée	Avec Pull-Up. (X_{IN} est l'image du niveau présent sur la broche)
1	X	X	Sortie	Sortie symétrique Push-Pull. X est le niveau délivré par la broche.

La lecture du registre PINx donne l'état des broches du PORTx orientées en entrée. L'écriture dans le registre PORTx fixe l'état des broches configurées en sortie. (PINx n'est accessible qu'en lecture)



La mise en œuvre des ports parallèles nécessite en premier **la programmation du registre DDRx** pour définir si les broches du circuit sont en entrée ou en sortie.

Schéma électrique associé à une broche du port A



Exercice :

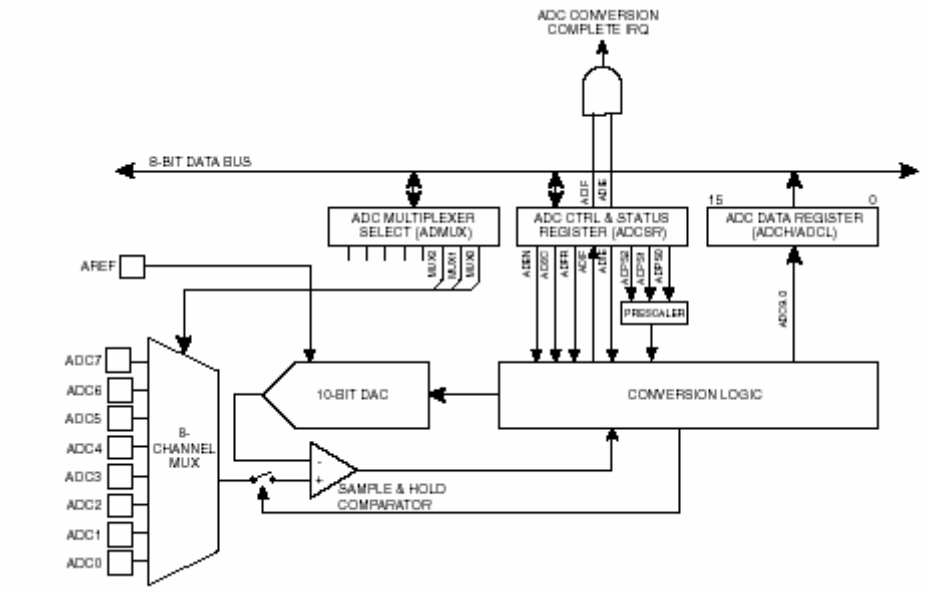
A partir du programme ci-dessous, vérifier sur le schéma électrique l'adéquation entre les valeurs des registres et les fonctions réalisées.

Exemple de programme d'initialisation des ports B et D:

```
        ser     temp            ; temp=FF
        out     DDRB,temp       ; port B en sortie
        clr     temp            ; temp=0
        out     DDRD,temp       ; port D en entrée
        ser     temp
        out     PORTD,temp      ; pull-up du port D actifs
```

5.7.3) Le convertisseur analogique-numérique

Architecture :



Voir la documentation en annexe pour les registres de configuration et de commande.

Exemple de programme de conversion:

```
init:
        ldi     temp,low (RAMEND)
        out     SPL,temp                ; initialisation pointeur pile poids faibles
        ldi     temp,high (RAMEND)
        out     SPH,temp                ; initialisation pointeur pile poids forts
        clr     temp
        out     ddra,temp               ; port A en entrée
        ldi     temp, 0x86              ; div 64 et CAN activé
        out     adcsr,temp
        clr     temp
        out     admux,temp              ; selection  voie 0 du CAN
convert:
        sbi     adcsr,adsc              ; lance conversion (bit 6 de adcsr à 1)
at_fin:
        in      temp,adcsr              ;teste le
        bst     temp,adif               ; bit
```

```
        brtc    at_fin                  ; 6 de adcsr et revient à at_fin si 0
        in      r_conv_l, adcl          ; lit le résultat de la conversion poids faibles
        in      r_conv_h, adch          ; idem poids Forts
        sbi     adcsr,adif              ; raz bit 6 (par écriture d'un 1)
        rcall   delai                   ; appelle une tempo
        rjmp    convert                 ; relance la conversion
```

5.7.4) Les Timers

Les timers peuvent être utilisés pour générer des temporisations précises, compter des évènements, mesurer des périodes ou fréquences de signaux, générer des signaux PWM …

Le programme ci-dessous donne un exemple d'utilisation du Timer 1 (on se reportera à la documentation en annexe pour la structure et les registres).

```
; ---------------------------------------------------------------------------------
; Sous-Programme de temporisation de 20ms (78*256µs) Quartz à 4MHZ;
; ---------------------------------------------------------------------------------

tempo_20:
        ldi temp,$0D
        out tccr1b,temp             ;démarrage compteur et clk/1024 (T=256µs)
tempo_20_1:
        in temp, tifr
        sbrs temp, ocf1a            ; test si flag ocf1a à 1
        rjmp tempo_20_1
        clr temp
        out tccr1b,temp             ;arrêt du compteur (pas d'horloge)
        ori temp,$10
        out tifr,temp               ;raz drapeau
        ret

; ----------------------------
Initialisation du Timer 1
; ----------------------------

init_timer1:
        ldi temp, $00
        out tccr1a, temp            ; timer en mode normal, pas d'action sur OC1
        out ocr1ah, temp            ; registre ocr1ah à 0
        ldi temp,78                 ;registre ocr1al à 78 (ou $4E)
        out ocr1al, temp            ; 78 = valeur de comparaison
        ret
```

5.7.5) l'UART (Unité Asynchrone de Réception Transmission)

C'est une interface série asynchrone. Elle permet de relier une unité centrale à des périphériques tels que :

- consoles de visualisation
- oscilloscopes ou autres appareils de mesure
- claviers
- souris
- modems
- …

ou à d'autres ordinateurs. Elle a l'avantage de relier deux systèmes sur une longueur de plusieurs dizaines de mètres mais elle a l'inconvénient d'être lente. Elle est de plus en plus remplacée par la liaison USB.

Exemple de programme mettant en œuvre la liaison série asynchrone en émission :
Nota : voir architecture et registres en annexe.

```
debut:
        mov data,car_1          ; le caractère à envoyer (car_1) est mis dans « data »
        rcall env_car           ; appel au sous-programme d'envoi caractère
        rcall delai             ; attente
        rjmp debut

-----------------------------------------------------
;Sous-programme d'initialisation de l'UART ;
-----------------------------------------------------
init_uart:
        ldi temp, $08           ;uart en émission
        out ucr,temp
        ldi temp,25             ; 9600 bauds si  Quartz à 4Mhz
        out ubrr,temp
        ret

-------------------------------------------------
;Sous-programme d'envoi du caractère ;
-------------------------------------------------
env_car:
        sbi usr,6               ; raz bit émission
        out udr,data            ; envoi le caractère
att_fin:
        in temp,usr             ; teste si caractère émis
        sbrs temp,6
        rjmp att_fin
        ret                     ; retour au programme appelant
```