Assembleur Microcontrôleur ATMEGA32

Les microcontrôleurs de la famille **ATMEGA** en technologie **CMOS** sont des modèles à 8 bits **AVR** basés sur l'architecture **RISC**. En exécutant des instructions dans un cycle d'horloge simple, **l'ATMEGA** réalise des opérations s'approchant de 1 **MIPS** par **MHZ** permettant de réaliser des systèmes à faible consommation électrique et simple au niveau électronique.

Note : Une partie des figures de ce document provienne de la documentation officielle **ATMEL** et plus spécialement du modèle **ATMEGA32**. Le reste est de ma conception personnel.

Table des Matières

Assembleur Microcontrôleur ATMEGA32 1
Introduction 7
La Syntaxe Utilisée 8
Le Plan Mémoire 9
La mémoire programme 9
La mémoire de donnée 10
La mémoire morte 10
Base et Opération Logiques 11
Les Opérations Booléennes 11
L'opérateur ET (&) 11
L'opérateur OU (!) 11
L'opérateur OU Exclusif (⊕) 11
L'opérateur NON (¯) 11
Les Bases 16, 10 et 2 12
Les Registres d'Etat SREG 13
Registre SREG (*Status Register*) 13
Mode d'Adressage 14
Les Opérateurs 14
Accès Direct Simple (Rd) 14
Accès Direct à deux opérateurs (Rr et Rd) 15
Adresse des Entrée/Sorties (A) 15
Adresse Direct des Données (k) 15
Adressage Indirect (X, Y ou Z) 16
Adressage Indirect avec Déplacement (q) 16
Adressage Indirect avec pré-décrément (-X, -Y ou -Z) 16
Adressage Indirect avec post-incrément (-X, -Y ou -Z) 17
Adressage des instructions LPM, ELPM et SPM 17
Adressage des instructions post-incrémentées LPM Z+, ELPM Z+ 18
Adressage Direct pour les instructions JMP et CALL 18
Adressage Indirect pour les instructions IJMP et ICALL 18
Adressage Relatif pour les instructions RJMP et RCALL 19
Les instructions de branchement 19
Jeu d'Instruction 20
Légende du jeu d'instruction 24
Registres et Opérateurs 24
RAMPX, RAMPY, RAMPZ, RAMPD 24
EIND 24
STACK 24
DRAPEAU 24
Les Instructions 25

Mode de Lecture 25
ADC – Add with Carry – Addition avec Retenue 25
ADD – Add without Carry – Addition sans Retenue 26
ADIW – Add Immediat Word – Addition Immédiate sur 16 bits 26
AND – Logical AND – ET Logique 27
ANDI – Logical AND with Immediate – ET Logique Immédiat 27
ASR – Arithmetic Shift Right – Décalage Arithmétique à Droite 28
BCLR – Bit Clear in SREG – Efface le Bit 's' Correspondant dans SREG 28
BLD – Bit Load from the T Flag in SREG to a bit b– Charge Registre avec T 29
BRBC – Branch if Bit in SREG is Cleared – Branche Conditionnelle Relatif si SREG(s)=0 29
BRBS – Branch if Bit in SREG is Set - Branche Conditionnelle Relatif si SREG(s) = 1 30
BRCC – Branch if Carry Cleared – Branchement si Pas de Retenue C = 0 30
BRCS – Branch if Carry Set - Branchement si Retenue C = 1 31
BREAK – Break – Pause du MCU 31
BREQ – Branch if Equal – Branchement si Egal Z = 1 32
BRGE – Branch if Greater or Equal (Signed) – Branchement si Supérieur ou Egal S=0 32
BRHC – Branch if Half Carry Flag is Cleared – Branchement si Demi Retenue H = 0 33
BRHS – Branch if Half Carry Flag is Set – Branchement si Demie Retenue H = 1 33
BRID – Branch if Global Interrupt is Disabled – Branchement si Interruption Arrêté 34
BRIE – Branch if Global Interrupt is Enabled – Branchement si Interruption Active 34
BRLO – Branch if Lower (Unsigned) – Branchement si Inférieur C = 1 35
BRLT – Branch if Less Than (Signed) – Branchement si Inferieur Signé S = 1 35
BRMI – Branch if Minus – Branchement si Négatif N = 1 36
BRNE – Branch if Not Equal – Branchement si non Egale Z = 0 36
BRPL – Branch if Plus – Branchement si Positif N = 0 37
BRSH – Branch if Same or Higher (Unsigned) – Branchement si = ou Plus Grand C=0 37
BRTC – Branch if the T Flag is Cleared – Branchement si Test T = 0 38
BRTS – Branch if the T Flag is Set – Branchement si Test T = 1 38
BRVC – Branch if Overflow Cleared – Branchement si non Dépassement V = 0 39
BRVS – Branch if Overflow Set – Branchement si Dépassement V = 1 39
BSET – Bit Set in SREG – Mise à 1 du Bit 's' de SREG 40
BST – Bit Store fromBit in Register to T Flag in SREG – Stock Registre dans T 40
CALL – Long Call to a Subroutine – Appel Long de Sous-programme 41
CBI – Clear Bit in I/O Register – Mise à 0 d'un Registre d'Entrée/Sortie 41
CBR – Clear Bits in Register – Efface les Bits d'un Registre en Complément à 1 42
CLC – Clear Carry Flag – Efface la Retenue C = 0 42
CLH – Clear Half Carry Flag – Efface la Demi Retenue H = 0 43
CLI – Clear Global Interrupt Flag – Désactive les Interruptions I = 0 43
CLN – Clear Negative Flag – Efface Négative N = 0 44
CLR – Clear Register – Effacement du Registre 44
CLS – Clear Signed Flag – Efface le Signe S = 0 45
CLT – Clear T Flag – Efface le Test T = 0 45
CLV – Clear Overflow Flag – Efface le Débordement V = 0 45
CLZ – Clear Zero Flag – Efface le Zéro Z = 0 46

COM – One's Complement – Complément à 1 46
CP – Compare – Comparaison de Registre 47
CPC – Compare with Carry – Comparaison avec Retenue 16 bits 47
CPI – Compare with Immediate – Comparaison avec Valeur Immédiate 48
CPSE – Compare Skip if Equal – Comparaison et Saut si Egale 48
DEC – Decrement - Décrémentation 49
EICALL – Extended Indirect Call to Subroutine – Appel Indirect au Sous-Programme 50
EIJMP – Extended Indirect Jump – Saut Indirect avec Z & EIND 50
ELPM – Extended Load Program Memory – Chargement Programme Mémoire Etendue 51
EOR – Exclusive OR – OU exclusif 52
FMUL – Fractional Multiply Unsigned – Multiplication Fractionnaire non Signé 52
FMULS – Fractional Multiply Signed – Multiplication Fractionnaire Signé 53
FMULSU – Fractional Multiply Signed with Unsigned – Multipication Signé-non Signé 54
ICALL – Indirect Call to Subroutine – Appel de Sous-programme Indirect 55
IJMP – Indirect Jump – Saut Indirect 55
IN - Load an I/O Location to Register – Lecture d'une Entrée/Sortie 56
INC – Increment - Incrément 56
JMP – Jump - Saut 57
LD – Load Indirect from Data Space to Register using Index X – Charge Indirect X 58
LD (LDD) – Load Indirect from Data Space to Register using Index Y – Charge Indirect Y 59
LD (LDD) – Load Indirect From Data Space to Register using Index Z – Charge Indirect Z 60
LDI – Load Immediate – Charge Valeur Immédiate 61
LDS – Load Direct from Data Space – Charge Direct 61
LPM – Load Program Memory – Charge un Programme en Mémoire 62
LSL – Logical Shift Left – Décalage Logique à Gauche 63
LSR – Logical Shift Right – Décalage Logique à Droite 63
MOV – Copy Register – Copie de Registre 64
MOVW– Copy Register Word – Copie d'un Registre sur 16 bits 64
MUL – Multiply Unsigned – Multiplication non Signé 65
MULS – Multiply Signed – Multiplication Signé 65
MULSU – Multiply Signed with Unsigned – Multiplication Signé et non Signé 66
NEG – Two's Complement – Complément à Deux 67
NOP – No Operation – Pas d'Opération 67
OR – Logical OR – OU Logique 68
ORI – Logical OR with Immediate – OU Logique Immédiat 68
OUT – Store Register to I/O Location – Ecriture d'une Entrée/Sortie 69
POP – Pop Register from Stack – Restaure la Donnée de la Pile 69
PUSH – Push Register on Stack – Stock un Registre sur la Pile 70
RCALL – Relative Call to Subroutine – Appel Relatif d'un Sous-programme 70
RET – Return from Subroutine – Retour de Sous-programme 71
RETI – Return from Interrupt – Retour d'Interruption 71
RJMP – Relative Jump – Saut Relatif 72
ROL – Rotate Left trough Carry – Rotation à Gauche avec Retenue 72
ROR – Rotate Right through Carry – Rotation à Droite avec Retenue 73

SBC – Subtract with Carry – Soustraction avec Retenue 74
SBCI – Subtract Immediate with Carry – Soustraction Immédiate avec Retenue 74
SBI – Set Bit in I/O Register – Active le Bit des Entrée/Sortie 75
SBIC – Skip if Bit in I/O Register is Cleared – Saute si Bit I/O est à 0 75
SBIS – Skip if Bit in I/O Register is Set – Saute si Bit I/O est à 1 76
SBIW – Subtract Immediate from Word – Soustraction Immédiate 16 bits 76
SBR – Set Bits in Register – Active le Bit du Registre 77
SBRC – Skip if Bit in Register is Cleared – Saute si le Bit du Registre est à 0 77
SBRS – Skip if Bit in Register is Set – Saute si le Bit du Registre est à 1 78
SEC – Set Carry Flag – Active la Retenue C = 1 78
SEH – Set Half Carry Flag – Active la Demi Retenue H = 1 79
SEI – Set Global Interrupt Flag – Active les Interruptions I = 1 79
SEN – Set Negative Flag – Active le Signe S = 1 79
SER – Set all Bits in Register – Active Tous les Bits du Registre 80
SES – Set Signed Flag – Active les Nombres Signés S = 1 80
SET – Set T Flag – Active le Test T = 1 81
SEV – Set Overflow Flag – Active le débordement V = 1 81
SEZ – Set Zero Flag – Active le Zéro Z = 1 81
SLEEP – Mode Sommeil 82
SPM – Store Program Memory – Stock le Programme en Mémoire 82
ST – Store Indirect From Register to Data Space using Index X – Stock Indirect X 84
ST (STD) – Store Indirect From Register to Data Space using Index Y – Stock Indirect Y 85
ST (STD) – Store Indirect From Register to Data Space using Index Z – Stock Indirect Z 86
STS – Store Direct to Data Space – Stock Direct dans l'Espace Donnée 87
SUB – Subtract without Carry – Soustraction sans Retenue 87
SUBI – Subtract Immediate – Soustraction Immediate 88
SWAP – Swap Nibbles – Echange Réciproque de Demi Octet 88
TST – Test for Zero or Minus – Test Zéro ou Négatif 89
WDR – Watchdog Reset – Remise à Zéro du Chien de Garde 89
L'assembleur *90*
Début Rapide 90
Démarrage 90
Assemblage de Votre Premier Fichier 90
Découverte et Correction d'Erreurs 91
Réassemblage 91
Assembler un programme source 92
Example 92
Directives d'Assemblage 93
BYTE – Reserve bytes to a variable - Réserve d'octet pour une variable 93
CSEG – Code Segment - Segment de Code 94
DB-Define constant byte(s) in program memory or EEPROM memory - Définit le(s) octet(s) constant 94
DEF - Set a symbolic name on a register - Définit un nom symbolique pour un registre 94
DEVICE – Define which device to assemble for - Définit le modèle pour l'assemblage 95
DSEG – Data Segment - Segment de Données 95

DW - Define constant word(s) in program memory or EEPROM memory - Définit le(s) mot(s) constant 95
ENDMACRO – End macro - Fin macro 96
EQU - Set a symbol equal to an expression - Attribut un symbole à une expression 96
ESEG – EEPROM Segment - Segment EEPROM 96
EXIT - Exit this file - Sortie du fichier 96
INCLUDE – Include another file - Importe la source d'un autre fichier 97
LIST - Turn the listfile generation on - Met en route la génération de listfile 97
LISTMAC – Turn macro expansion on - Met en route l'extension macro 97
MACRO – Begin macro - Commence une macro 98
NOLIST - Turn listfile generation off - Arrêt de la génération de listfile 98
ORG - Set program origin - Origine du programme en mémoire 98
SET - Set a symbol equal to an expression - Déclare un symbole pour une expression 99
Les Expressions de l'Assembleur 100
Operandes 100
Fonctions 100
Opérateurs 100
Unaire Symbole : ! 100
Négation Symbole : ~ 100
Moins Symbole : - 100
Multiplication Symbole : * 100
Division Symbole : / 101
Addition Symbole : + 101
Soustraction Symbole : - 101
Décalage Gauche Symbole : << 101
Décalage Droit Symbol: >> 101
Plus Petit Que Symbole : < 101
Plus Petit ou Equale Symbole : <= 101
Plus Grand Que Symbole : > 101
Plus Grand ou Equale Symbole : >= 101
Equale Symbole : == 101
Non Equale Symbole : != 101
ET entre valeur Symbol : & 102
OU Exclusif Logique Symbol : ^ 102
OU entre deux valeur Symbol : | 102
ET Logique Symbol : && 102
OU Logique Symbol : || 102

Introduction

Ce document est la suite du document Microcontrôleur **ATMEL ATMEGA** que j'ai réalisé en novembre 2003.

Vous trouverez dans ce document plusieurs chapitres sur l'initiation au microcontrôleur **ATMEGA** et son assembleur avec le descriptif complet des 131 instructions.

Tout d'abord, une présentation des éléments du microcontrôleur est nécessaire pour comprendre le fonctionnement du **MCU** et de l'action des instructions sur les éléments tel que la mémoire, l'unité arithmétique et logique, **l'EEPROM** et les interfaces diverses.

Un rappel sommaire des opérations logiques (**ET**, **OU**, **NON**, **OU Exclusif**) et la conversion décimale hexadécimale et binaire sera aborder.

Le fonctionnement des indicateurs du registre de statut **SREG** avec le positionnement en fonction des opérations arithmétiques et logiques.

Le mode d'adressage et les opérations de saut et sous-programme et enfin le détaille de chaque instruction est prévue suivie par la programmation en assembleur avec quelques exemples plus ou moins complexe.

Puis le desciptif des 131 instructions suivie d'une entrée en matière de l'assembleur **d'ATMEL** clotura ce document.

Bien entendu, vous pouvez me signaler les erreurs éventuelles qui se seraient glissées dans ce document à l'issus de mon plein gré ! Mon mail est maintenant : Balade.nono@voila.fr.

Très bonne lecture à vous ! Jean-Noël, en Janvier 2005 en Gambie.

La Syntaxe Utilisée

Pour que ce document ne soit pas trop indigeste, je l'ai organisé en chapitre et découpé en module simple ou chaque instruction est détaillé et en prenant des exemples simple puis complexe pour entrée en douceur, si c'est réellement possible, dans l'univers du microcontrôleur.

Les fonctions des registres sont marquées en **GRAS** et les instructions sont en ***GRAS ITALIQUE***.

Pour les chiffres :

Les données en décimales sont écritent sans signe spécifique,
Les données en Hexadécimales sont précédé du signe '$', exemple : $10 vaut 16 en décimal,
Les données en binaire sont précédé d'un b et suivie de 8 '0/1', exemple : b00010001 soit 17 en décimal.

Le Plan Mémoire

Le cœurs **AVR** combine 32 registres spéciaux travaillants directement avec l'Unité Arithmétique de Logique **ALU**, qui représente le registre d'accumulateur **A** (**B** ou **D**) dans les microcontrôleurs classiques.

Ces registres spéciaux permettent à deux registres indépendants d'être en accès directe par l'intermédiaire d'une simple instruction et exécutée sur un seul cycle d'horloge. Cela signifie que pendant un cycle d'horloge simple l'Unité Arithmétique et Logique **ALU** exécute l'opération et le résultat est stocké en arrière dans le registre de sortie, le tout dans un cycle d'horloge. L'architecture résultante est plus efficace en réalisant des opérations jusqu'à dix fois plus rapidement qu'avec des microcontrôleurs conventionnels **CISC**.

Les registres spéciaux sont dit aussi registre d'accès rapide et 6 des 32 registres peuvent être employés comme trois registre d'adresse 16 bits pour l'adressage indirects d'espace de données (**X, Y & Z**). Le troisième **Z** est aussi employé comme indicateur d'adresse pour la fonction de consultation de table des constantes.

Les 32 registres sont détallés dans le tableau qui suit avec l'adresse effective dans la mémoire **SRAM** :

Bit 7 à 0	Adresse	Registre Spéciaux
R0	$00	
R1	$01	
Rn	$xx	
R26	$1A	Registre **X** Partie Basse
R27	$1B	Registre **X** Partie Haute
R28	$1C	Registre **Y** Partie Basse
R29	$1D	Registre **Y** Partie Haute
R30	$1E	Registre **Z** Partie Basse
R31	$1F	Registre **Z** Partie Haute

Comme vous l'avez compris, trois types de mémoire sont utilisées dans la série **ATMEGA**, la mémoire programme **FLASH**, la mémoire de donnée **SRAM** et la mémoire morte de type **EEPROM**.

La mémoire programme

La mémoire programme permet de stoker et de faire fonctionner le microcontrôleur, il contient de 4 à 256 Ko de programme selon le modèle du microcontrôleur. Le nombre d'écriture sur cette mémoire est limité à 10.000, largement suffisant pour la majorité des applications. La figure 1 donne un exemple de l'adressage de la mémoire **FLASH** du modèle **ATMEGA 32**.

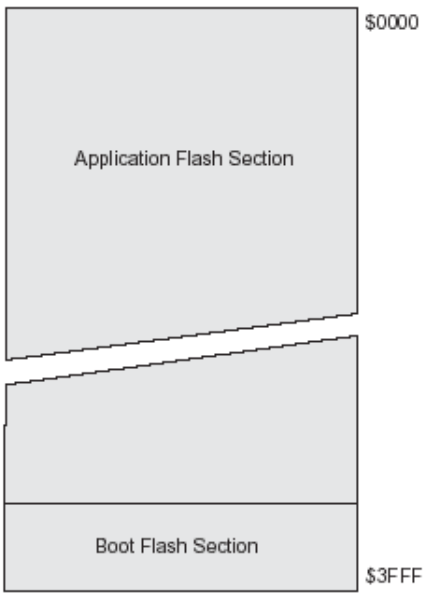


Figure 1, Adressage de la mémoire **FLASH**.

La mémoire de donnée

La mémoire de donnée contient les 32 registres de travail, les 64 registres de commande et la mémoire **SRAM** pour les variables du programme de 2048 octets pour le modèle **ATMEGA 32**. La figure 2 présente les relations entre espace physique et registre.

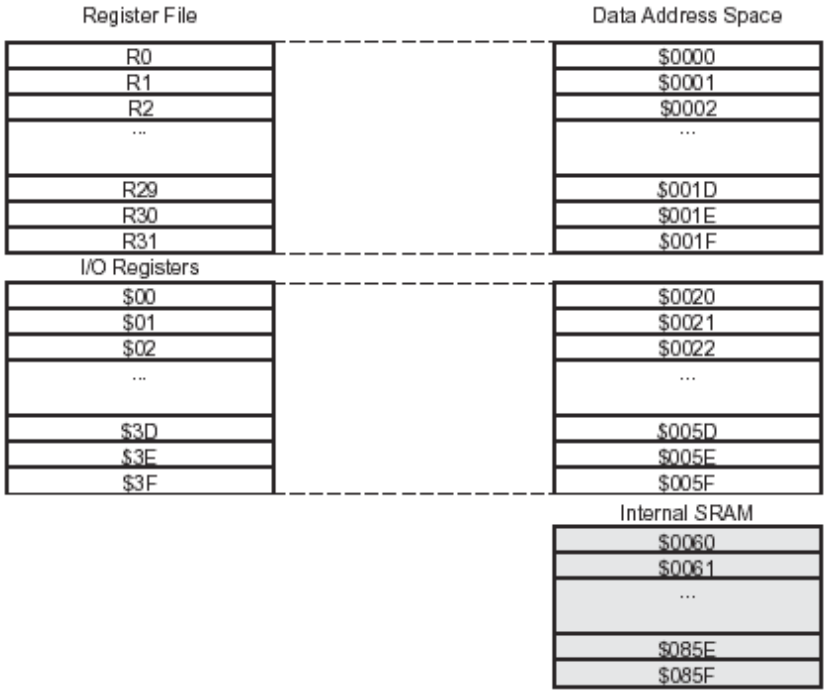


Figure 2, la mémoire de donnée avec la **SRAM**.

La mémoire morte

La mémoire morte est de type **EEPROM** d'accès plus complexe contiendra la configuration du programme et les données importantes qui seront sauvées pendant l'absence de courant électrique. On peut écrire jusqu'à 100.000 fois dans l'**EEPROM**. La taille de **l'EEPROM** est fonction du modèle de microcontrôleur **ATMEGA** (de 256 bits à 8 Ko).

Base et Opération Logiques

Voici un petit rappel sur les bases hexadécimale, binaire et les opérations logiques pour en comprendre le fonctionnement.

Attention : les opérateurs logiques ici employés sont des références internationnales, mais ne sont maleureusement pas les mêmes que ceux utilisés par l'assembleur pour des raisons de droit principalement.

Les Opérations Booléennes

L'opérateur ET (&)

L'opérateur **ET** est une multiplication booléenne qui si les deux valeurs sont à 1 donnera un résultat à 1 :

Opérateur 1	Opérateur 2	ET Logique
0	0	0
0	1	0
1	0	0
1	1	1

L'opérateur OU (!)

L'opérateur **OU** est une addition booléenne qui si l'une des deux valeurs est à 1 donnera un résultat à 1 :

Opérateur 1	Opérateur 2	OU Logique
0	0	0
0	1	1
1	0	1
1	1	1

L'opérateur OU Exclusif (Å)

L'opérateur **OU Exclusif** est une addition booléenne exclusive qui si l'une des deux valeurs seulement est à 1 donnera un résultat à 1 :

Opérateur 1	Opérateur 2	OU Exc. Logique
0	0	0
0	1	1
1	0	1
1	1	0

L'opérateur NON (¯)

L'opérateur **NON** est une négation booléenne qui inverse la valeur, un 1 donnera un résultat à 0 et vis versa :

Opérateur	NON Logique
0	1
1	0

La combinaison des opérations est possible et par exemple un **ET** et un **NON** donnera un **NONET** qui inversera le résultat tout simplement. Dans le document qui suit, les valeurs NON seront misent en rouge.

Les Bases 16, 10 et 2

Les chiffres et les nombres ne sont pas représentés tel quel dans la mémoire du microcontrôleur, ils sont converties en information binaire (en base 2) sur 8 bits (Bit = élément binaire valant 0 ou 1), qui donne un octet (octet ensemble de 8 bits) et sont représentées de la manière suivant : b00000000. On peut avoir jusqu'à 256 valeurs différente avec se système, de 0 à 255 avec 8 bits d'informations.

Inconvénient majeur, ce n'est pas très pratique d'aligner des lignes de 0 et de 1 pour écriture un programme. La conversion en hexadécimale (base 16) à donc été prise comme référence pour les systèmes à microcontrôleur et à microprocesseur à 8 bits et plus. Il reste que le binaire est pratique pour le contrôle des Port d'entrée/sortie ou pour localiser rapidement une information dans un registre.

La table qui suit donne un exemple de conversion simple et rapide :

Décimale	Binaire	Hexadécimale
0	b0000	$0
1	b0001	$1
2	b0010	$2
3	b0011	$3
4	b0100	$4
5	b0101	$5
6	b0110	$6
7	b0111	$7
8	b1000	$8
9	b1001	$9
10	b1010	$A
11	b1011	$B
12	b1100	$C
13	b1101	$D
14	b1110	$E
15	b1111	$F

Comme on peut le voir, l'hexadécimal ou encore 'l'hexa' est constitué de chiffre et de lettre, avec un chiffre en hexa on représente 4 bits, il faut donc 2 chiffres hexa pour faire un octet, celui de gauche sera le poids fort du nombre et celui de droite sera le poids faible :

Décimal	Poids Fort	Poids Faible	Hexa
165	$A	$5	$A5
37	$2	$5	$25

Le signe de l'hexa est le dollar '$' et le signe du binaire est le 'b', il n'y a pas de signe pour le décimal.

Pour convertir un chiffre de l'hexa en décimale, il suffit de multiplier par 16 le poids fort et d'ajouter le poids faible : $A = 10, et 10 x 16 = 160 + $5 = 165, vraiment simple non.

Pour faire l'inverse c'est un peu plus dur, il faut diviser le nombre décimal par 16 qui donnera le poids fort et le reste de la division donnera le poids faible.

165	16
10	5

Ceci reste valable pour les valeurs inférieures à 255 bien sûr.

Pour les valeurs supérieures, il faut au préalable diviser le nombre par 256 et reprendre le même principe pour les deux parties, d'autre méthode existe, mais je me limiterais à ce rappel qui est suffisant dans la majorité des cas.

Les Registres d'Etat SREG

Le registre indispensable au système pour fonctionner est le registre d'état **SREG**. En effet, ce registre permet de réaliser les opérations de branchements qui autorisent des applications complexes.

Registre SREG (*Status Register*)

Le registre **SREG** ou registre d'état sert principalement avec les fonctions arithmétiques et logiques pour les opérations de branchements. Il indique et autorise aussi le fonctionnement des interruptions. Il est modifié par le résultat des manipulations mathématiques et logiques. C'est le principal registre qui sert à effectuer les branchements conditionnels après une opération arithmétique ou logique. Il peut être lu et écrit à tout moment sans aucune restriction.

Adresse	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
$3F	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E

I ***Global Interrupt Enable*** sert à activer (1) ou interdire (0) toutes les sources d'interruptions. Si ce bit n'est pas activé alors que vous avez programmé des interruptions, elles ne seront pas prises en compte.

T ***Copy Storage*** joue un rôle de tampon lors de manipulation de bits avec les instructions ***BLD*** et ***BST***.

H ***Half Cary*** signale qu'une demie-retenue a été réalisé lors de l'emploi d'une instruction arithmétique.

S ***Sign bit*** bit de signe résultant d'un **OU** exclusif avec le bit **N** et **V**.

V ***Overflow bit*** indique un dépassement de capacité lors des opérations arithmétique et logique.

N ***Negative bit*** signale que le résultat de la dernière manipulation arithmétique est négatif.

Z ***Zéro bit*** le résultat de la dernière manipulation arithmétique est égal à zéro.

C ***Cary bit*** l'opération arithmétique à donné lieu à une retenue.

Et bien maintenant, voici comment cela marche avec quelques exemples simples.

Mode d'Adressage

Le microcontrôleur **ATMEGA** soutient des modes d'adressage puissants et efficaces pour l'accès à la mémoire programme **FLASH** et la mémoire de données **SRAM**, les registres et les Entrée/Sorties.

Cette section décrit les modes d'adressage soutenus par l'ATMEGA. **OP** signifie la partie de code d'**OPération** du mot d'instruction. Pour simplifier, toutes les figures montrent l'emplacement exact des bits d'adressage. Pour généraliser, les termes **RAMEND** et **FLASHEND** ont été employés pour représenter l'emplacement le plus haut dans l'espace des données et de programme, respectivement.

Note : Tous les modes d'adressage ne sont pas présents dans tous les modèles. Vérifiez que le modèle que vous voulez utiliser possède l'instruction dans le tableau résumé de la documentation Anglaise, le programme de compilation vous indiquera une erreur sur l'instruction est manquante dans le modèle grace à la directive **DEVICE** (voir en fin de document).

Les Opérateurs

Les opérateurs permettent de choisir la source et la destination de l'opération qui sera réalisé par le microcontrôleur dans le code instruction. Ces opérations peuvent travailler sur le registre d'accès rapide (**ALU**), les registres d'Entrée/Sortie, la **RAM** (**SRAM**) pour les données, ou la mémoire **FLASH** pour les constantes.

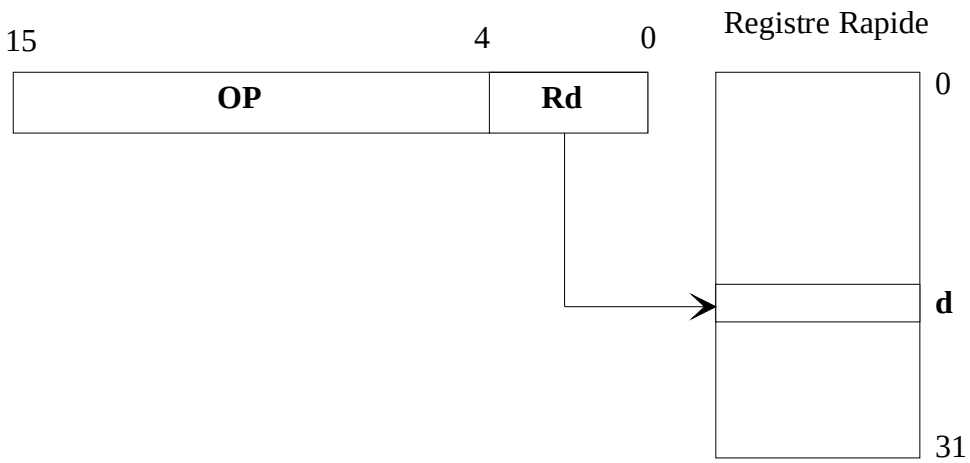
Voici un récapitulatif des différents opérateurs utilisés dans la documentation qui suit :

Rd : Registre Rapide de Destination (ou source) écrit,
Rr : Registre Rapide Source lu,
A : Adresse Registre d'Entrée/Sortie,
k : Adresse des Données en **SRAM**,
b : Bit de sélection dans un octet,
X, Y, Z : Registre d'Adresse Indirect (X=R27:R26, Y=R29:R28 et Z=R31:R30),

Nous allons maintenant expliquer le fonctionnement interne du microcontrôleur avec ces **opérateurs**.

Accès Direct Simple (Rd)

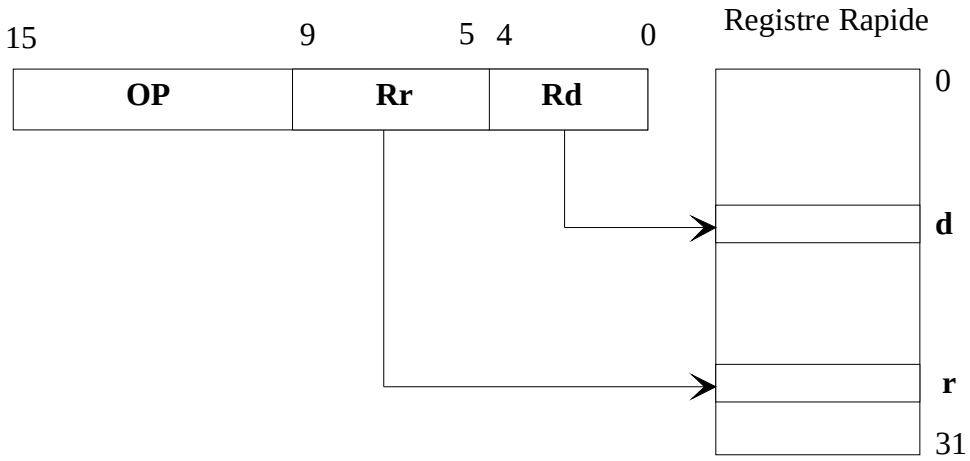
L'Opérateur **Rd** permet de choisir l'un des 32 registres d'accès rapides celui que l'on utilisera dans la mémoire.



L'opérateur est donc contenu dans la valeur **Rd** qui est codé sur 5 bits (32 solutions).

Accès Direct à deux opérateurs (Rr et Rd)

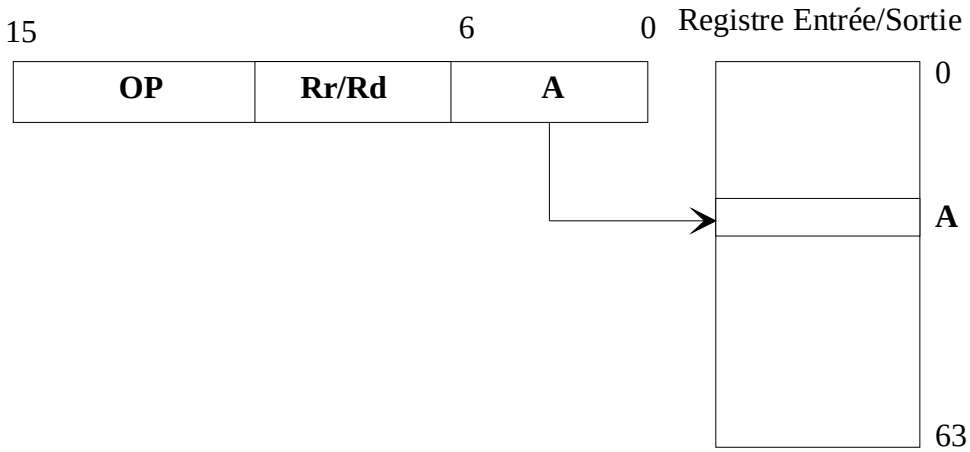
L'accès direct avec les deux opérateurs **Rr** et **Rd** permet de travailler avec deux registres d'accès rapides en même temps, le résultat sera placé dans le registre **Rd** (Destination).



Le résultat sera dans **Rd**. L'opérateur **Rr** comme l'opérateur **Rd** est codé sur 5 bits (32 solutions).

Adresse des Entrée/Sorties (A)

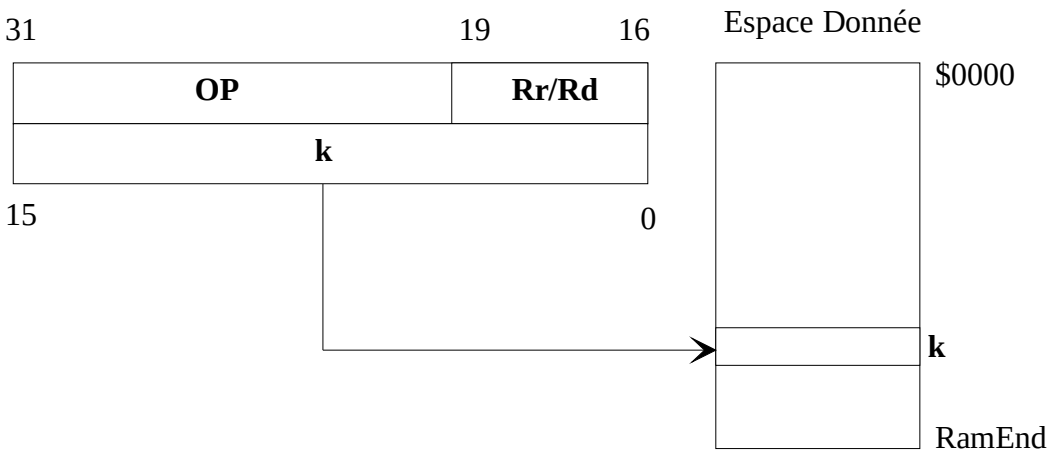
L'adressage des 64 registres d'Entrée/Sortie est réalisé par l'opérateur **A** comme suit :



L'adresse du registre d'Entrée/Sortie est codée sur 6 bits, soit 64 positions possibles.

Adresse Direct des Données (k)

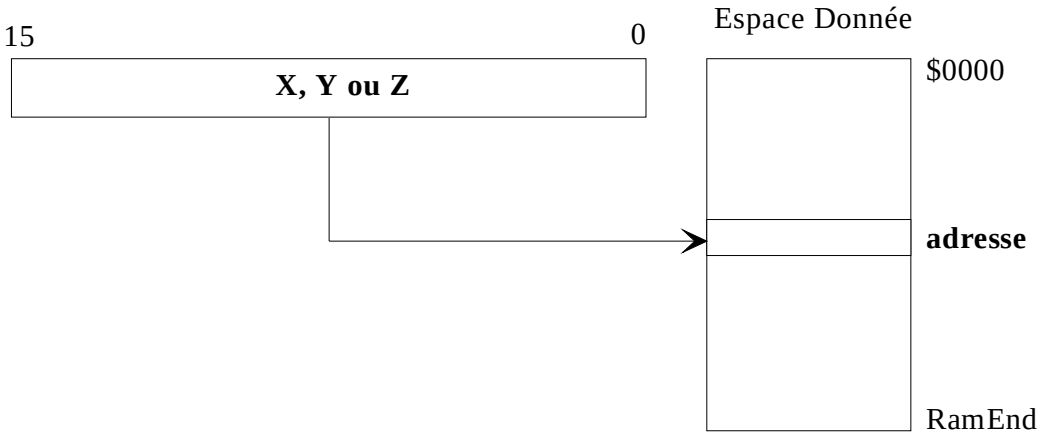
L'adressage direct des données est réalisé par l'utilisation de deux mots à 16 bits comme le montre la figure suivante :



L'adresse **k** sur 16 bits est placée sur deux octets successifs.

Adressage Indirect (X, Y ou Z)

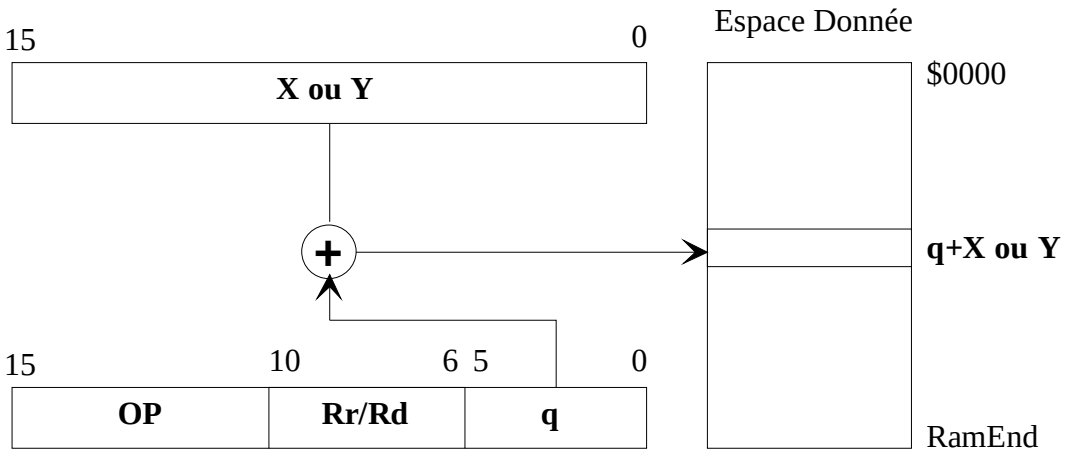
L'adressage indirect permet de charger l'opérateur avec le contenue de l'adresse pointé par les registres **X**, **Y** ou **Z** à 16 bits.



L'adresse de la **SRAM** est totalement accessible.

Adressage Indirect avec Déplacement (q)

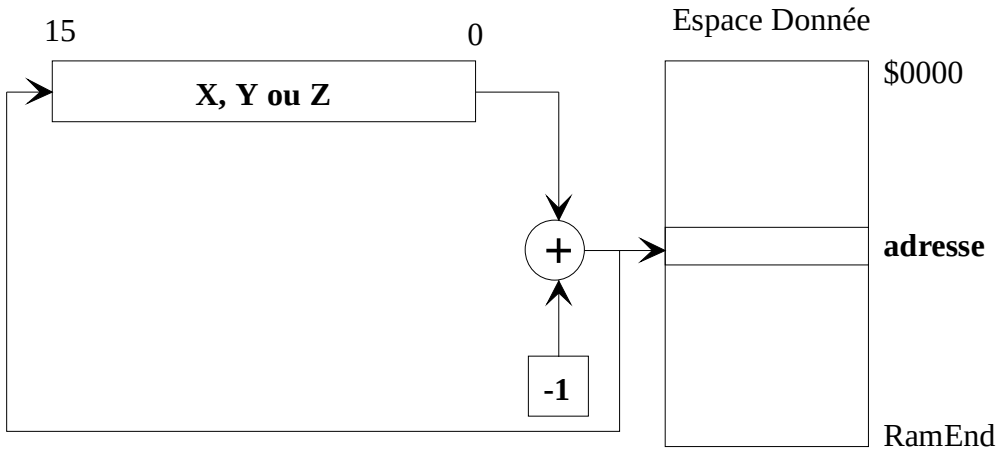
L'adressage indirect permet d'utiliser le registre **X** ou **Y** comme référence d'adresse et **q** comme valeur ajouté pour le déplacement, l'adresse sera donc la somme de **X** ou **Y** et **q** :



La valeur de **q** est codée sur 6 bits.

Adressage Indirect avec pré-décrément (-X, -Y ou -Z)

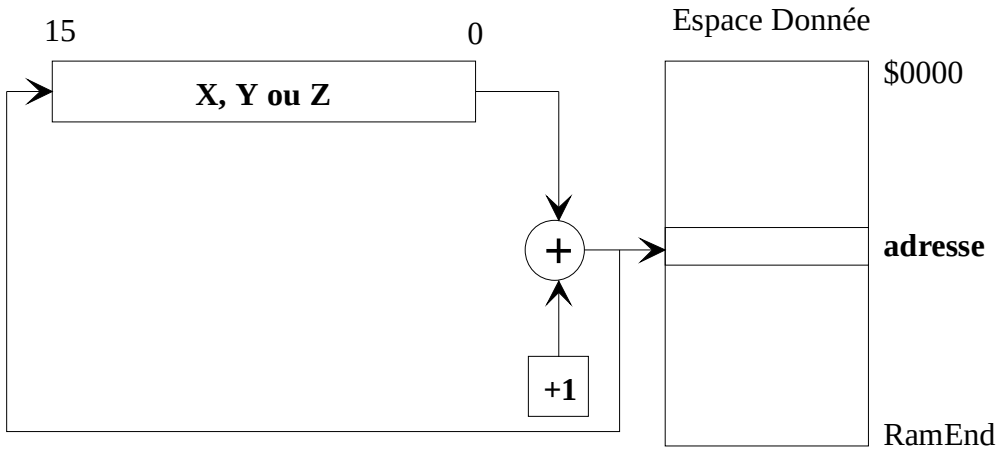
L'adressage indirect avec pré-décrémentation permet de lire une suite de donnée dans une table en partant du haut de la mémoire :



Le registre **X**, **Y** ou **Z** est diminué de 1 à chaque utilisation, donc l'adresse finale sera diminuée de 1.

Adressage Indirect avec post-incrément (-X, -Y ou -Z)

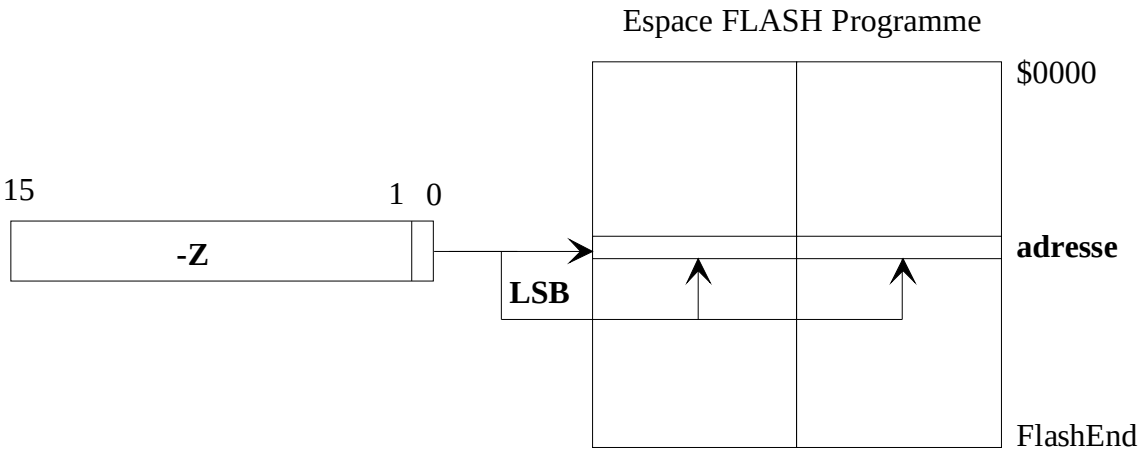
L'adressage indirect avec post-incrémentation permet de lire une suite de donnée dans une table en partant du bas de la mémoire :



Le registre **X**, **Y** ou **Z** est augmenté de 1 à chaque utilisation, donc l'adresse finale sera augmentée de 1 à chaque fois.

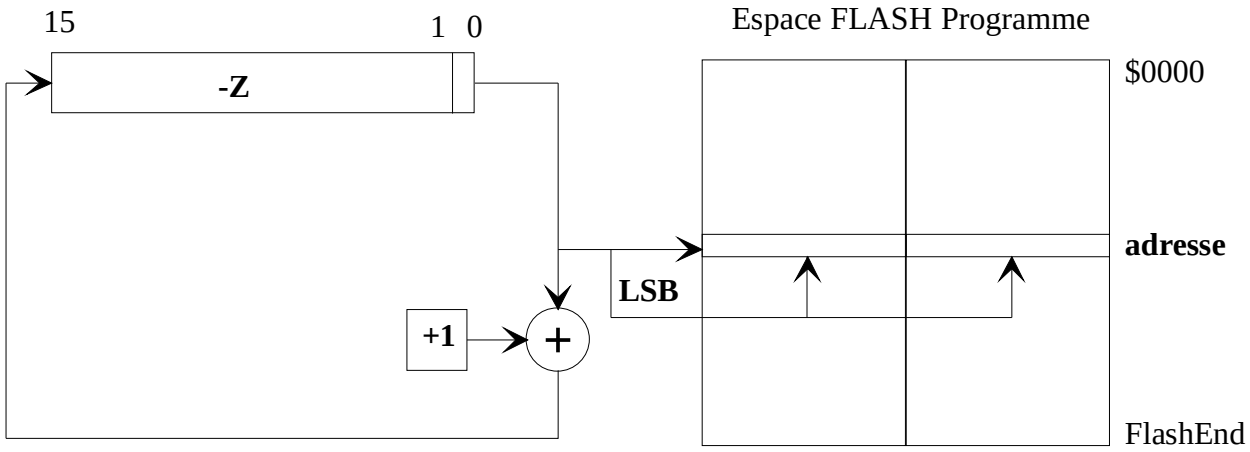
Adressage des instructions LPM, ELPM et SPM

L'adressage de la mémoire programme est spécifié par le contenu du registre **Z**. Les 15 bits de poids fort choisissent l'adresse du mot. Pour **LPM**, le **LSB** (poids faible) choisit l'octet bas si à 0 ou l'octet haut si à 1. Pour **SPM**, le **LSB** doit être à 0. Si **ELPM** est employé, le Registre de **RAMPZ** est employé pour étendre le registre **Z** à 21 bits.



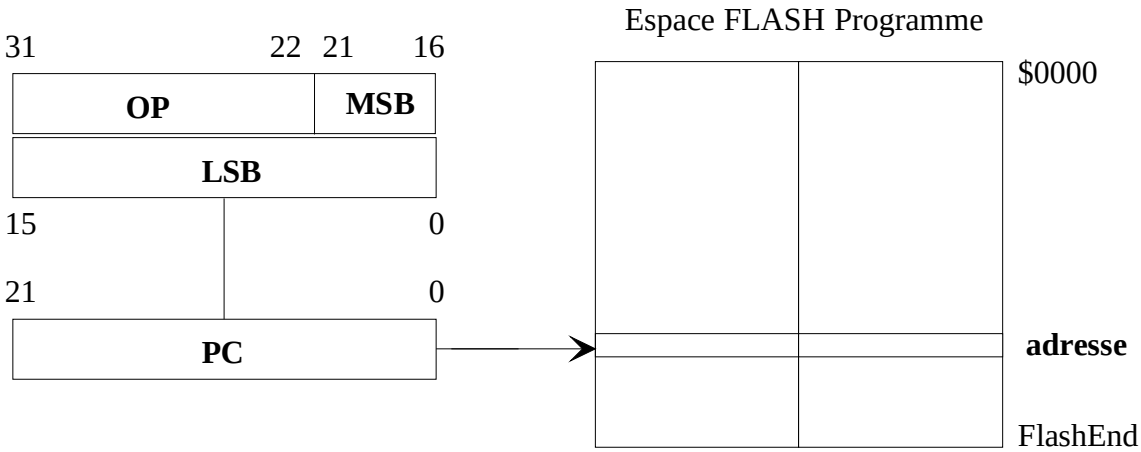
Adressage des instructions post-incrémentées LPM Z+, ELPM Z+

L'adressage de la mémoire programme est spécifié par le contenu du registre **Z**. Les 15 bits de poids fort choisissent l'adresse du mot. Pour **LPM Z+**, le **LSB** (poids faible) choisit l'octet bas si à 0 ou l'octet haut si à 1. Si **ELPM Z+** est employé, le Registre de **RAMPZ** est employé pour étendre le registre **Z** à 21 bits.



Adressage Direct pour les instructions JMP et CALL

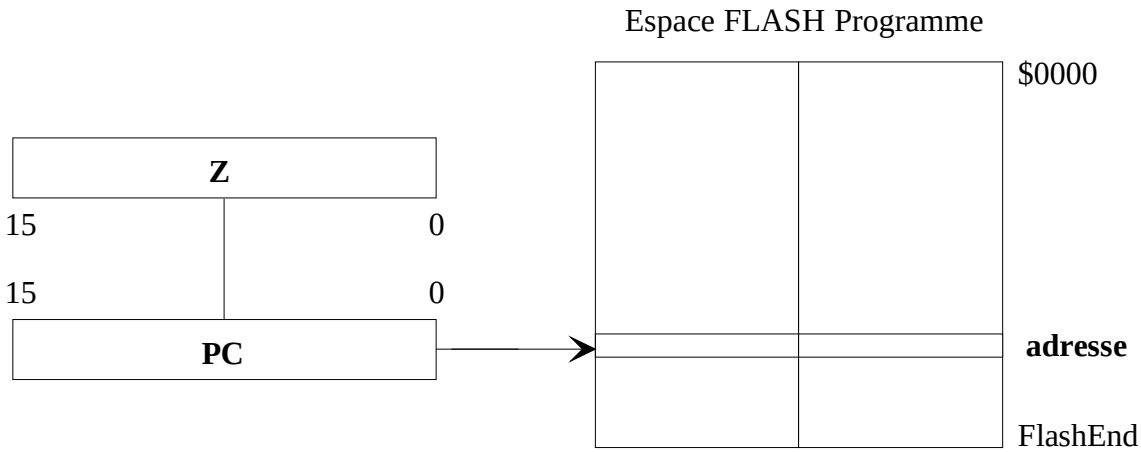
L'adressage direct va modifier le **PC** (Pointeur Programme) pour les sauts et l'appel des sous-programmes :



Le **PC** est constitué des 16 bits **LSB** et des 6 bits **MSB** pour créer les 22 bits du pointeur, l'espace est au maximum de 4 096 **Ko** pour le futur ! La suite de l'exécution du programme sera faite sur l'adresse du **PC**.

Adressage Indirect pour les instructions IJMP et ICALL

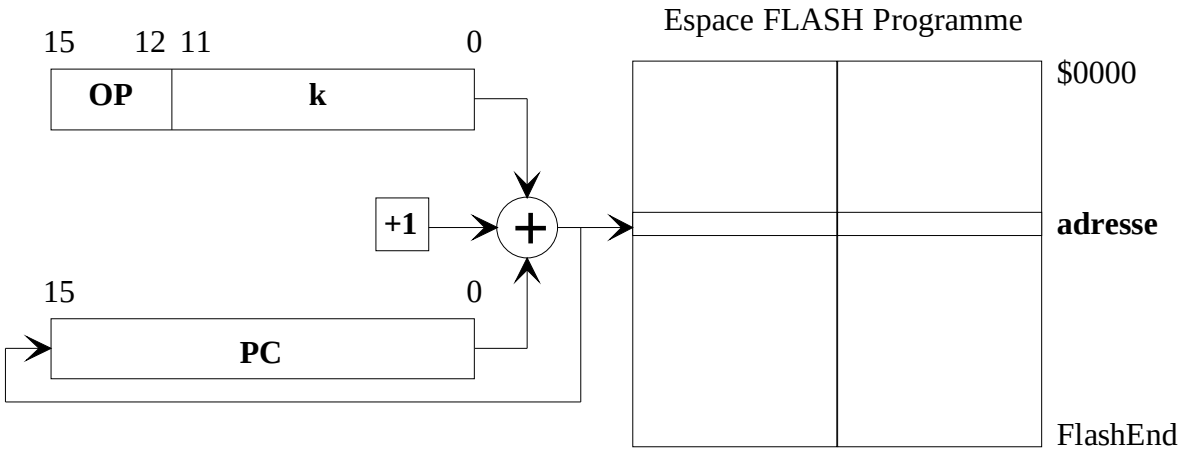
L'adressage indirect avec le registre **Z** comme référence pour modifier le **PC** :



L'exécution de la prochaine instruction sera effectuée à l'adresse du registre **Z** qui est transféré dans le **PC**.

Adressage Relatif pour les instructions RJMP et RCALL

L'adressage relatif permet de continuer un programme après un saut relatif à la valeur de **k** qui sera cumulé au **PC**. La valeur de **k** est comprise entre -2048 et +2047 et le **PC** sera égale à : **PC** = **k** + 1:



Il est important de bien utiliser les étiquettes d'adressage pour cette instruction. (Voir les branchements dans le chapitre de l'assembleur).

Les instructions de branchement

Les instructions de branchements permettent de tester des grandeurs entre elle :

Évaluez	Booléen	Instruction	Remarque
Rd > Rr	Z & (N ⊕ V) = 0	BRLT	Signé
Rd >= Rr	(N ⊕ V) = 0	BRGE	Signé
Rd = Rr	Z = 1	BREQ	Signé
Rd <= Rr	Z ! (N ⊕ V) = 1	BRGE	Signé
Rd < Rr	(N ⊕ V) = 1	BRLT	Signé
Rd > Rr	C ! Z = 0	BRLO	Non signé
Rd >= Rr	C = 0	BRCC	Non signé
Rd = Rr	Z = 1	BREQ	Non signé
Rd <= Rr	C ! Z = 1	BRSH	Non signé
Rd < Rr	C = 1	BRLO	Non signé
Retenu	C = 1	BRCS	Simple
Négatif	N = 1	BRMI	Simple
Déborde	V = 1	BRVS	Simple
Zéro	Z = 1	BREQ	Simple

Nous verrons dans le détail de chaque instruction des exemples concrets.

Jeu d'Instruction

L'ATMEGA à un jeu de 131 instructions qui seront détaillées dans la suite de ce document.

Le code instruction est à utiliser dans les programmes écrit en assembleur, l'opérant spécifi les variables ou constantes utilisés par l'instruction, le registre **SREG** est modifier par l'instruction en fonction du résultat de celle-ci, et le nombre de cycle **CPU** est donnée en dernier.

Les instructions sont présentées dans les tableaux qui suivent par type :

Code	Opérant	Description	Opération	Registre	Cycle
Instructions Arithmétiques et Logique					
ADD	Rd, Rr	Addition sans Retenue	Rd = Rd + Rr	Z C N V S H	1
ADC	Rd, Rr	Addition avec Retenue	Rd = Rd + Rr + C	Z C N V S H	1
ADIW	Rd, k	Addition Immédiat 16 bits	Rd+1:Rd = Rd+1:Rd + k	Z C N V S	2 (1)
SUB	Rd, Rr	Soustraction sans Retenue	Rd = Rd – Rr	Z C N V S H	1
SUBI	Rd, k	Soustraction Immédiat sans Retenue	Rd = Rd – k	Z C N V S H	1
SBC	Rd, Rr	Soustraction avec Retenue	Rd = Rd – Rr – C	Z C N V S H	1
SBCI	Rd, K	Soustraction Immédiat avec Retenue	Rd = Rd – k – C	Z C N V S H	1
SBIW	Rd, K	Soustraction Immédiat 16 bits	Rd+1:Rd = Rd+1:Rd – k	Z C N V S	2 (1)
AND	Rd, Rr	ET Logique	Rd = Rd & Rr	Z N V S	1
ANDI	Rd, k	ET Logique Immédiat	Rd = Rd & k	Z N V S	1
OR	Rd, Rr	OU Logique	Rd = Rd ! Rr	Z N V S	1
ORI	Rd, k	OU Logique Immédiat	Rd = Rd ! k	Z N V S	1
EOR	Rd, Rr	OU Exclusif	Rd = Rd ⊕ Rr	Z N V S	1
COM	Rd	Complément à 1	Rd = $FF – Rd	Z C N V S	1
NEG	Rd	Négation (Complément à 2)	Rd = $00 – Rd	Z C N V S	1
SBR	Rd, k	Mise à 1 dans Registre (OU)	Rd = Rd ! k	Z N V S	1
CBR	Rd, k	Mise à 0 dans Registre (ET)	Rd = Rd & ($FF - k)	Z N V S	1
INC	Rd	Incrément	Rd = Rd + 1	Z N V S	1
DEC	Rd	Décrément	Rd = Rd – 1	Z N V S	1
TST	Rd	Test à Zéro ou Négatif	Rd = Rd ! Rd	Z N V S	1
CLR	Rd	Effacement du Registre	Rd = $00	Z N V S	1
SER	Rd	Mis à 1 du Registre	Rd = $FF	-	1
MUL	Rd, Rr	Multiplication non Signé	R1:R0 = Rd x Rr (UU)	Z C	2 (1)
MULS	Rd, Rr	Multiplication Signé	R1:R0 = Rd x Rr (SS)	Z C	2 (1)
MULSU	Rd, Rr	Multiplication Signé avec non Signé	R1:R0 = Rd x Rr (SU)	Z C	2 (1)
FMUL	Rd, Rr	Multiplication Fractionnaire non Signé	R1:R0=(Rd x Rr)<<1 (UU)	Z C	2 (1)
FMULS	Rd, Rr	Multiplication Fractionnaire Signé	R1:R0=(Rd x Rr)<<1(SS)	Z C	2 (1)
FMULSU	Rd, Rr	Multiplication Fractionnaire Signé avec non Signé	R1:R0=(Rd x Rr)<<1(SU)	Z C	2 (1)

Code	Opérant	Description	Opération	Registre	Cycle
Instructions de Branchement et Saut					
RJMP	k	Saut Relatif	PC = PC + k + 1	-	2
IJMP		Saut Indirect Z	PC(15:0)=Z, PC(21:16)=0	-	2 (1)
EIJMP		Saut Etendu Indirect Z	PC(15:0)=Z, PC(21:16)=EIND	-	2 (1)
JMP	k	Saut	PC = k	-	3 (1)
RCALL	k	Sous-Programme Relatif	PC = PC + k + 1	-	3/4 (4)
ICALL		Sous-Programme Indirect Z	PC(15:0)=Z, PC(21:16)=0	-	3/4(1,4)
EICALL		Sous-Programme Etendu Indirect Z	PC(15:0)=Z, PC(21:16)=EIND	-	4 (4)
CALL	k	Sous-Programme	PC = k	-	4/5(1,4)
RET		Retour de Sous-Programme	PC = STACK	-	4/5 (4)
RETI		Retour d'Interruption	PC = STACK	I	4/5 (4)
CPSE	Rd, Rr	Compare et Saute si Égal	Si Rd=Rr PC=PC+2 ou 3	-	1/2/3
CP	Rd, Rr	Comparer	Rd – Rr	Z C N V S H	1
CPC	Rd, Rr	Comparez avec Retenue	Rd - Rr - C	Z C N V S H	1
CPI	Rd, k	Comparez Immédiat	Rd – K	Z C N V S H	1
SBRC	Rr, b	Sautez si le bit du Registre à 0	Si Rr(b)=0 PC=PC+2 ou 3	-	1/2/3
SBRS	Rr, b	Sautez si le bit du Registre à 1	Si Rr(b)=1 PC=PC+2 ou 3	-	1/2/3
SBIC	A, b	Sautez si le bit du Registre I/O à 0	Si A, b=0 PC=PC+2 ou 3	-	1/2/3
SBIS	A, b	Sautez si le bit du Registre I/O à 1	Si A, b =1 PC=PC+2 ou 3	-	1/2/3
BRBS	s, k	Branche si SREG(s) à 1	Si SREG(s)=1 PC=PC+k+1	-	1/2
BRBC	s, k	Branche si SREG(s) à 0	Si SREG(s)=0 PC=PC+k+1	-	1/2
BREQ	k	Branche si Égal Z à 1	Si Z=1 PC=PC+k+1	-	1/2
BRNE	k	Branche si Non Égal Z à 0	Si Z=0 PC=PC+k+1	-	1/2
BRCS	k	Branche si Retenue C à 1	Si C=1 PC=PC+k+1	-	1/2
BRCC	k	Branche si Retenue C à 0	Si C = 0 PC=PC+k+1	-	1/2
BRSH	k	Branche si Egal ou Plus Haut	Si C=0 PC=PC+k+1	-	1/2
BRLO	k	Branche si Plus bas	si C = 1 PC=PC+k+1	-	1/2
BRMI	k	Branche si Négatif	Si N=0 PC=PC+k+1	-	1/2
BRPL	k	Branche si Positif	Si N = 0 PC=PC+k+1	-	1/2
BRGE	k	Branche si Plus Grand ou Égal Signé	Si N⊕V=0 PC=PC+k+1	-	1/2
BRLT	k	Branche si Moins Que Signé	Si N⊕V=1 PC=PC+k+1	-	1/2
BRHS	k	Branche si Drapeau Moitié H à 1	Si H=1 PC=PC+k+1	-	1/2
BRHC	k	Branche si Drapeau Moitié H à 0	Si H=0 PC=PC+k+1	-	1/2
BRTS	k	Branche si Drapeau T=1	Si T=1 PC=PC+k+1	-	1/2
BRTC	k	Branche si Drapeau T=0	Si T=0 PC=PC+k+1	-	1/2
BRVS	k	Branche si Débordement V=1	Si V=1 PC=PC+k+1	-	1/2
BRVC	k	Branche si Débordement v=0	Si V=0 PC=PC+k+1	-	1/2
BRIE	k	Branche si Interruption active I=1	Si I=1 PC=PC+k+1	-	1/2
BRID	k	Branche si Interruption inactive I=0	Si I=0 PC=PC+k+1	-	1/2

Code	Opérant	Description	Opération	Registre	Cycle
Instructions de Transfert de Donnée					
MOV	Rd, Rr	Déplacement	Rd = Rr	-	1
MOVW	Rd, Rr	Déplacement 16 bits	Rd+1:Rd = Rr+1:Rr	-	1 (1)
LDI	Rd, k	Charge Immédiat	Rd = k	-	1
LDS	Rd, k	Charge Directe en Mémoire	Rd = (k)	-	2 (1,4)
LD	Rd, X	Charge Indirect	Rd = (X)	-	2 (1,4)
LD	Rd, X+	Charge Indirect Post Incrément	Rd = (X) et X = X + 1	-	2 (1,4)
LD	Rd, -X	Charge Indirect Pré Décrément	X = X – 1 et Rd = (X)	-	2 (1,4)
LD	Rd, Y	Charge Indirect	Rd = (Y)	-	2 (1,4)
LD	Rd, Y+	Charge Indirect Post Incrément	Rd = (Y) et Y = Y + 1	-	2 (1,4)
LD	Rd, -Y	Charge Indirect Pré Décrément	Y = Y – 1 et Rd = (Y)	-	2 (1,4)
LDD	Rd, Y+q	Charge Indirect avec Déplacement	Rd = (Y + q)	-	2 (1,4)
LD	Rd, Z	Charge Indirect	Rd = (Z)	-	2 (2,4)
LD	Rd, Z+	Charge Indirect Post Incrément	Rd = (Z) et Z = Z+1	-	2 (2,4)
LD	Rd, -Z	Charge Indirect Pré Décrément	Z = Z – 1 et Rd = (Z)	-	2 (2,4)
LDD	Rd, Z+q	Charge Indirect avec Déplacement	Rd = (Z + q)	-	2 (1,4)
STS	k, Rr	Stock Direct en Mémoire	(k) = Rd	-	2 (1,4)
ST	X, Rr	Stock Indirect	(X) = Rr	-	2 (2,4)
ST	X+, Rr	Stock Indirect Post Incrément	(X) = Rr et X = X + 1	-	2 (2,4)
ST	-X, Rr	Stock Indirect Pré Décrément	X = X – 1 et (X) = Rr	-	2 (2,4)
ST	Y, Rr	Stock Indirect	(Y) = Rr	-	2 (2,4)
ST	Y+, Rr	Stock Indirect Post Incrément	(Y) = Rr et Y = Y + 1	-	2 (2,4)
ST	-Y, Rr	Stock Indirect Pré Décrément	Y = Y – 1 et (Y) = Rr	-	2 (2,4)
STD	Y+q,Rr	Stock Indirect avec Déplacement	(Y + q) = Rr	-	2 (1,4)
ST	Z, Rr	Stock Indirect	(Z) = Rr	-	2 (2,4)
ST	Z+, Rr	Stock Indirect et Incrément postal	(Z) = Rr et Z = Z + 1	-	2 (2,4)
ST	-Z, Rr	Stock Indirect et Pré décroissance	Z = Z – 1 et (Z) = Rr	-	2 (2,4)
STD	Z+q,Rr	Stock Indirect avec Déplacement	(Z + q) = Rr	-	2 (1,4)
LPM		Mémoire Programme de Charge	R0 = (Z)	-	3 (3)
LPM	Rd, Z	Mémoire Programme de Charge	Rd = (Z)	-	3 (3)
LPM	Rd, Z+	Mémoire Programme de Charge Post Incrément	Rd = (Z) et Z = Z + 1	-	3 (3)
ELPM	Extended	Mémoire Programme de Charge	R0 = (RAMPZ:Z)	-	3 (1)
ELPM	Rd, Z	Mémoire Programme de Charge Etendue	Rd = (RAMPZ:Z)	-	3 (1)
ELPM	Rd, Z+	Mémoire Programme de Charge Etendue Post Incrément	Rd=(RAMPZ:Z) et Z=Z + 1	-	3 (1)
SPM		Stock Mémoire Programme	(Z) = R1:R0	-	- (1)
IN	Rd, A	Entrée d'un Port	Rd = I/O(A)	-	1
OUT	A, Rr	Sortie sur Port	I/O(A) = Rr	-	1
PUSH	Rr	Entrée du Registre de Pile	STACK = Rr	-	2 (1)
POP	Rd	Sortie du Registre de Pile	Rd = STACK	-	2 (1)

Code	Opér.	Description	Opération	Registre	Cycle
Instructions de Bit et Bit test					
LSL	Rd	Décalage Logique à Gauche	C=Rd(7), Rd(n+1)=Rd(n), Rd(0)=0	Z C N V H	1
LSR	Rd	Décalage Logique à Droite	C=Rd(0), Rd(n)=Rd(n+1), Rd(7)=0	Z C N V	1
ROL	Rd	Rotation à Gauche	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z C N V H	1
ROR	Rd	Rotation à Droite	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z C N V	1
ASR	Rd	Décalage Arithmétique à Droite	Rd(n) = Rd(n+1), n=0:6	Z C N V	1
SWAP	Rd	Échange demi-parti Réciproque	Rd(3:0)=Rd(7:4) et Rd(7:4)=Rd(0:3)	-	1
BSET	s	Drapeau SREG(s) à 1	SREG(s) = 1	SREG(s)	1
BCLR	s	Drapeau SREG(s) à 0	SREG(s) = 0	SREG(s)	1
SBI	A, b	Registre d'entrée-sortie à 1	I/O(A, b) = 1	-	2
CBI	A, b	Registre d'entrée-sortie à 0	I/O(A, b) = 0	-	2
BST	Rr, b	Drapeau T = position b du Registre	T = Rr(b)	T	1
BLD	Rd, b	Registre position b = Drapeau T	Rd(b) = T	-	1
SEC		Drapeau Retenue à 1	C = 1	C	1
CLC		Drapeau Retenue à 0	C = 0	C	1
SEN		Drapeau Négatif à 1	N = 1	N	1
CLN		Drapeau Négatif à 0	N = 0	N	1
SEZ		Drapeau Zéro à 1	Z = 1	Z	1
CLZ		Drapeau Zéro à 0	Z = 0	Z	1
SEI		Drapeau d'Interruption Permit	I = 1	I	1
CLI		Drapeau d'Interruption Arrêté	I = 0	I	1
SES		Drapeau Signé à 1	S = 1	S	1
CLS		Drapeau Signé à 0	S = 0	S	1
SEV		Drapeau Débordement à 1	V = 1	V	1
CLV		Drapeau Débordement à 0	V = 0	V	1
SET		Drapeau Transfert à 1	T = 1	T	1
CLT		Purifiez Transfert à 0	T = 0	T	1
SEH		Drapeau Moitié de Retenue H à 1	H = 1	H	1
CLH		Drapeau Moitié de Retenue H à 0	H = 0	H	1

Code	Opérant	Description	Registre	Cycle
Instruction de Contrôle MCU				
BREAK		Voir la description de l'instruction de BREAK	-	1 (1)
NOP		Pas d'Opération (Attente d'un cycle)	-	1
SLEEP		Voir la description de l'instruction de SLEEP	-	1
WDR		Voir le chapitre sur le Watchdog ou l'instruction WDR	-	1

Notes : 1. Cette instruction n'est pas disponible dans tous les modèles.

2. L'ensemble des variantes des instructions n'est pas disponible dans tous les modèles.

3. L'ensemble des variantes de l'instruction **LPM** n'est pas disponible dans tous les modèles.

4. Le cycle d'horloge pour les accès mémoire de Données n'est pas valable pour des accès via l'interface externe de **RAM**. Pour **LD**, **ST**, **LDS**, **STS**, **PUSH**, **POP**, ajouter un cycle de plus pour chaque état d'attend. Pour **CALL**, **ICALL**, **EICALL**, **RCALL**, **RET**, **RETI** avec **PC** à 16 bits, l'ajoutent de trois cycles plus deux

cycles pour chaque état d'attente. Pour **CALL**, **ICALL**, **EICALL**, **RCALL**, **RET**, **RETI** avec le **PC** à 22 bits, l'ajoute de cinq cycles plus trois cycles pour chaque état d'attend.

Légende du jeu d'instruction

Registres et Opérateurs

Rd : Registre de Destination (ou de source) Ecrit,
Rr : Registre Source Lu,
k : Adresse en mémoire,
b : Bit du Registre ou Registre d'Entrée/Sortie (3 bits),
s : Bit du Registre de Statut (3 bits),
X, Y, Z : Registre d'Adresse Indirect (X=R27:R26, Y=R29:R28 et Z=R31:R30),
A : Adresse d'Entrée/Sortie,
q : Adressage Indirect (6 bits).

RAMPX, RAMPY, RAMPZ, RAMPD

Les registres sont enchaînés avec **X-, Y-** et **Z-** permettant l'adressage indirect de l'espace de données du **MCU** en débordant l'espace de données de 64 **Ko**.

Registre enchaîné **Z-** permettant l'adressage direct de l'espace de données entier du **MCU** en débordant l'espace de données de 64 **Ko**.

EIND

Registre enchaîné avec le mot d'instruction permettant un appel indirect sur l'espace programme entier du **MCU** en débordant l'espace programme de 64 **Ko**.

STACK

La pile pour le retour d'adresse des registres poussés dans **SP**, Indicateur de Pile pour empiler-dépiler.

DRAPEAU

= : Drapeau affecté par instruction
0 : Drapeau purifié par instruction
1 : Drapeau mis par instruction
- : Drapeau non affecté par instruction

Les Instructions

Voici le détail de chaque instruction.

Mode de Lecture

Pour lire rapidement ces fiches, il faut localiser en marron l'instruction suivie par la correspondance des lettres avec le mnémonique, suivie par un descriptif de l'instruction et une formule simple représentant le fonctionnement de l'instruction.

Ensuite, on précise la portée des variables, leurs champs d'application et les éventuelles limites.

Le code de l'instruction sur 16 bits ou plus si nécessaire.

Un tableau regroupant les informations importantes de l'instruction, le mnémonique, les opérateurs, le résultat, le PC, le nombre d'octet occupé, et le nombre de cycle MCU.

Les modification éventuelle du registre de statut SREG et les opérations logiques de chaque indicateur avec en rouge la négation de la valeur : R1 = Non R1. Les signes '**!**' = **OU**, '**&**' = **ET**, Å = **OU Exclusif**.

Pour finir des exemples simples.

ADC – Add with Carry – Addition avec Retenue

Ajout deux registres **Rd** & **Rr** avec le contenu de la retenue **C** et place le résultat dans **Rd,** soit :

Rd = Rd + Rr + C

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ADC	Rd, Rr	Rd	PC + 1	2	1

Code :

Code Instruction	0001	11rd	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = Rd3 & Rr3 ! Rr3 & R3 ! R3 & Rd3
S = N ⊕ V
V = Rd7 & Rr7 & R7 ! Rd7 & Rr7 & R7
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = Rd7 & Rr7 ! Rr7 & R7 ! R7 & Rd7

Exemple :

```
; Addition sur 16 bits
        Add     r2, r0          ; Addition bit de poids faible
        Adc     r3, r1          ; Addition avec retenue bit de poids fort
```

ADD – Add without Carry – Addition sans Retenue

Ajout deux registres **Rd** & **Rr** sans la retenue **C** et place le résultat dans **Rd** :

Rd = Rd + Rr

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ADD	Rd, Rr	Rd	PC + 1	2	1

Code :

Code Instruction	0000	11rd	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = Rd3 & Rr3 ! Rr3 & R3 ! R3 & Rd3
S = N ⊕ V
V = Rd7 & Rr7 & R7 ! Rd7 & Rr7 & R7
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = Rd7 & Rr7 ! Rr7 & R7 ! R7 & Rd7

Exemple :

```
; Addition sur 8 bit
        Add     r1,r2           ; Addition de r2 avec r1 (r1=r1+r2)
        Add     r28,r28         ; Addition de r28 avec lui même (r28=r28+r28)
```

ADIW – Add Immediat Word – Addition Immédiate sur 16 bits

Ajoute une valeur immédiate **k** (0 - 63) à une paire de registre et place le résultat de la paire de registre. Cette instruction fonctionne sur 4 paires de registre uniquement et est adapter aux opérations sur les registres d'indicateur. Cette instruction n'est pas disponible dans tous les modèles.

Rd+1:Rd = Rd+1:Rd + k

Avec : Rd = {24, 26, 28 ou 30} et **k** = 0 à 63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ADIW	Rd+1:Rd, k	Rd+1 :Rd	PC + 1	2	2

Code :

Code Instruction	1001	0110	kkdd	kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	X	X	X	X	X

S = N ⊕ V
V = Rdh7 & R15
N = R15
Z = R15&R14&R13&R12&R11&R10&R9 & R8 & R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = R15 & Rdh7

Exemple :

```
; Addition Immédiate
        Adiw    r25:24,1        ; Addition de 1 à r25:r24
        Adiw    ZH:ZL,63        ; Addition de 63 au registre Z (r31:r30)
```

AND – Logical AND – ET Logique

Exécute un **ET** logique entre le contenu de **Rd** et **Rr** et place le résultat dans **Rd** :

Rd = Rd & Rr

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
AND	Rd, Rr	Rd	PC + 1	2	1

Code :

Code Instruction	0010	00rd	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	**X**	**0**	**X**	**X**	-

S = N ⊕ V
V = 0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0

Exemple :

```
; Et logique
        and     r2, r3          ; Et entre r2 et r3, résultat dans r2
        ldi     r16,1           ; Charge r16 avec b00000001
        and     r2, r16         ; Isole le bit 0 dans r2
```

ANDI – Logical AND with Immediate – ET Logique Immédiat

Et logique entre le contenu de **Rd** et une constante **k** et place le résultat dans **Rd**.

Rd = Rd & k

Avec : Rd = 16 à 31 et **k** = 0 à 255.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ANDI	Rd, k	Rd	PC + 1	2	1

Code :

Code Instruction	0111	kkkk	dddd	kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	**X**	**0**	**X**	**X**	-

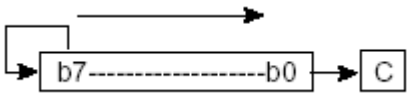
S = N ⊕ V
V = 0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0

Exemple :

```
; Et logique Immédiat
        andi    r17, $0F        ; Efface les 4 bits de poids fort de r17
        andi    r18, $10        ; Isole le bit 4 de r18
        andi    r19, $AA        ; Efface un bit sur deux de r19
```

ASR – Arithmetic Shift Right – Décalage Arithmétique à Droite

Décale tous les bits dans **Rd** d'une place à droit, le bit 7 est replacé dans le nouveau bit 7, le bit 0 est chargé dans l'indicateur **C** de **SREG**. Cette opération divise efficacement une valeur signée par deux sans changer son signe. Le registre **SREG** est modifié après le décalage.



Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ASR	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	1001	010d	dddd	0101

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	X	X	X	X	X

S = N ⊕ V (avec N et V après le décalage)
V = N ⊕ C (avec N et C après le décalage)
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = Rd0

Exemple :

```
; Décalage Arithmétique à Droite
        ldi     r16, $10        ; Charge la valeur décimale 16 dans r16
        asr     r16             ; r16 = r16 / 2
        ldi     r17, $FC        ; Charge -4 dans r17
        asr     r17             ; r17 = r17 / 2
```

BCLR – Bit Clear in SREG – Efface le Bit 's' Correspondant dans SREG

Efface la valeur de la position du bit **s** correspondant dans le registre de statut **SREG**, la valeur de **s** est donc la position du bit dans le sens poids fort – poids faible. Les autres bits ne sont pas modifiés.

SREG(s) = 0

Avec : s = 0 à 7.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BCLR	s	SREG	PC + 1	2	1

Code :

Code Instruction	1001	0100	1sss	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	X	X	X	X	X	X	X	X

I = 0 si s = 7 **T** = 0 si s = 6 **H** = 0 si s = 5 **S** = 0 si s = 4 **V** =0 si s = 3
N =0 si s = 2 **Z** =0 si s = 1 **C** =0 si s = 0

Exemple :

```
; Efface le bit s dans SREG
        bclr    0               ; Efface la retenue
        bclr    7               ; Désactive les interruptions
```

BLD – Bit Load from the T Flag in SREG to a bit b– Charge Registre avec T

Copie le drapeau **T** de **SREG** dans le registre **Rd** à la position du bit **b**.

Rd(b) = T

Avec : Rd = 0 à 31 et **b** = 0 à 7.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BLD	Rd, b	Rd	PC + 1	2	1

Code :

Code Instruction	1111	100d	dddd	0bbb

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Copie le T dans Rd à la position b et la fonction complémentaire
        bld     r0, 4           ; Charge T dans le bit 4 de r0
        bst     r1, 2           ; Stock le bit 2 de r1 dans T
```

BRBC – Branch if Bit in SREG is Cleared – Branche Conditionnelle Relatif si SREG(s)=0

Evalue le bit **s** dans **SREG** et branchement relatif au **PC** si le bit est à 0. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si SREG(s) = 0 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : s = 0 à 7 et **k** = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRBC	s, k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	01kk	kkkk	ksss

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si SREG(s) = 0
        cpi     r20, 5          ; Comparer r20 avec la valeur 5
        brbc    1, bitzero      ; Branche si Zéro = 0
        ...
bitzero:    nop                 ; Destination du branchement
```

BRBS – Branch if Bit in SREG is Set - Branche Conditionnelle Relatif si SREG(s) = 1

Evalue le bit **s** dans **SREG** et branchement relatif au **PC** si le bit est à 1. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si SREG(s) = 1 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : s = 0 à 7 et **k** = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRBS	s, k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	ksss

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si SREG(s) = 1
            bst     r0, 3           ; Charge r0 avec la valeur 3
            brbs    6, bitset       ; Branche si T = 1
            ...
bitset:     nop                     ; Destination du branchement
```

BRCC – Branch if Carry Cleared – Branchement si Pas de Retenue C = 0

Branchement conditionnel relatif si la retenue C est à 0. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si C = 0 alors PC . PC + k + 1, sinon PC . PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRCC	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	01kk	kkkk	K000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si C = 0
            Add     r22, r23        ; Addition r23 avec r22
            Brcc    nocarry         ; Branchement si la retenue C = 0
            ...
nocarry:    nop                     ; Destination du branchement
```

BRCS – Branch if Carry Set - Branchement si Retenue C = 1

Branchement conditionnel relatif si la retenue **C** est à 1. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si C = 1 alors PC . PC + k + 1, sinon PC . PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRCS	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	K000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si C = 1
        cpi     r26, $56        ; Compare r26 avec $56
        brcs    carry           ; Branchement si retenue à 1
        ...
carry:  nop                     ; Destination du branchement
```

BREAK – Break – Pause du MCU

L'instruction **BREAK** est employée par le système de mise au point et n'est pas normalement employée dans le logiciel d'application. Quand l'instruction **BREAK** est exécutée le **MCU** est arrêté. Cela donne accès au programme de mise au point et aux ressources internes. Si un bit de blocage est mis, ou **JTAGEN** ou **OCDEN** ne sont pas programmés, le **MCU** traitera l'instruction **BREAK** comme un **NOP** et n'entrera pas au mode Arrêté. Cette instruction n'est pas disponible sur tous les modèles.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BREAK	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0101	1001	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Non utilisé dans vos programmes
```

BREQ – Branch if Equal – Branchement si Egal Z = 1

Branchement conditionnel relatif si le drapeau Zéro **Z** est à 1. Si l'instruction est exécutée immédiatement après les instructions **CP**, **CCPP**, **SUB** ou **SUBI**, le branchement arrive si et seulement si **Rd** est égale **Rr** bit à bit. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si Rd = Rr (Z = 1) alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BREQ	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	k001

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si Egale (Z = 1)
        cp      r1, r0          ; Compare r1 avec r0
        breq    equal           ; Branchement si égale
        ...
equal:  nop                     ; Branchement de destination
```

BRGE – Branch if Greater or Equal (Signed) – Branchement si Supérieur ou Egal S=0

Branchement conditionnel relatif si le drapeau Signé **S** est à 0. Si l'instruction est exécutée immédiatement après les instructions **CP**, **CCPP**, **SUB** ou **SUBI**, le branchement arrive si et seulement si **Rd** signé est plus grand ou égal à **Rr** signé. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si Rd = Rr (N Å V = 0) alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRGE	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	01kk	Kkkk	k100

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si >= (S = 0)
        cp      r11, r12        ; Compare r11 et r12
        brge    greateq         ; Branche si r11 >= r12 (signé)
        ...
greateq: nop                    ; Branchement de destination
```

BRHC – Branch if Half Carry Flag is Cleared – Branchement si Demi Retenue H = 0

Branchement conditionnel relatif si le drapeau **H** est à 0. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si H = 0 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRHC	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	01kk	kkkk	k101

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si (H = 0)
        brhc    hclear          ; Branche si demi retenue est à 0
        ...
hclear: nop                     ; Branchement de destination
```

BRHS – Branch if Half Carry Flag is Set – Branchement si Demie Retenue H = 1

Branchement conditionnel relatif si le drapeau **H** est à 1. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si H = 1 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRHC	K	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	k101

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si (H = 1)
        brhs    hset            ; Branche si demie retenue = 1
        ...
hset:   nop                     ; Branchement de destination
```

BRID – Branch if Global Interrupt is Disabled – Branchement si Interruption Arrêté

Branchement conditionnel relatif si le drapeau **I** est à 0. Donc si les Interruptions sont désactivés le branchement à lieu. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si I = 0 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRID	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	01kk	kkkk	k111

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si Interruption désactivé (I = 1)
        brid    Intdis          ; Branche si Interruption désactivé = 0
        ...
Intdis: nop                     ; Branchement de destination
```

BRIE – Branch if Global Interrupt is Enabled – Branchement si Interruption Active

Branchement conditionnel relatif si le drapeau **I** est à 1. Donc si les Interruptions sont activés le branchement à lieu. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si I = 1 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRIE	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	k111

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si Interruption désactivé (I = 1)
        brie    Inten           ; Branche si Interruption activé = 1
        ...
Inten:  nop                     ; Branchement de destination
```

BRLO – Branch if Lower (Unsigned) – Branchement si Inférieur C = 1

Branchement conditionnel relatif si la retenue **C** non signé est à 1. Si l'instruction est exécutée immédiatement après les instructions **CP**, **CCPP**, **SUB** ou **SUBI**, le branchement arrive si et seulement si **Rd** non signé est plus petit que **Rr** non signé. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si Rd < Rr (C = 1) alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRLO	K	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	k000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si < (C = 1)
        cp      r11, r12        ; Compare r11 et r12
        brlo    inf             ; Branche si r11 < r12 (non signé)
        ...
Inf:    nop                     ; Branchement de destination
```

BRLT – Branch if Less Than (Signed) – Branchement si Inferieur Signé S = 1

Branchement conditionnel relatif si le drapeau Signé **S** est à 1. Si l'instruction est exécutée immédiatement après les instructions **CP**, **CCPP**, **SUB** ou **SUBI**, le branchement arrive si et seulement si **Rd** signé est plus grand ou égal à **Rr** signé. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si Rd < Rr (N Å V = 1) alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRLT	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	k100

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si < (S = 1)
        cp      r11, r12        ; Compare r11 et r12
        brlt    lees            ; Branche si r11 < r12 (signé)
        ...
lees:   nop                     ; Branchement de destination
```

BRMI – Branch if Minus – Branchement si Négatif N = 1

Branchement conditionnel relatif si la valeur est négative **N** à 1. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si N = 1 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRMI	K	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	k010

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si N = 1
            cp      r11, r12        ; Compare r11 et r12
            brmi    minus           ; Branche si N = 1
            ...
minus:      nop                     ; Branchement de destination
```

BRNE – Branch if Not Equal – Branchement si non Egale Z = 0

Branchement conditionnel relatif si non égale **Z** à 0. Si l'instruction est exécutée immédiatement après les instructions **CP**, **CCPP**, **SUB** ou **SUBI**, le branchement arrive si et seulement si **Rd** n'est pas égale à **Rr**. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si Rd ¹ Rr (Z = 0) alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRNE	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	01kk	kkkk	k001

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si ≠ (Z = 0)
            cp      r11, r12        ; Compare r11 et r12
            brne    negal           ; Branche si r11 ≠ r12 (non signé)
            ...
negal:      nop                     ; Branchement de destination
```

BRPL – Branch if Plus – Branchement si Positif N = 0

Branchement conditionnel relatif si la valeur est positive **N** à 0. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si N = 0 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRPL	K	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	01kk	kkkk	k010

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si N = 0
        cp      r11, r12        ; Compare r11 et r12
        brpl    plus            ; Branche si N = 0
        ...
plus:   nop                     ; Branchement de destination
```

BRSH – Branch if Same or Higher (Unsigned) – Branchement si = ou Plus Grand C=0

Branchement conditionnel relatif si égal ou supérieur **C** est à 0. Si l'instruction est exécutée immédiatement après les instructions **CP**, **CCPP**, **SUB** ou **SUBI**, le branchement arrive si et seulement si **Rd** non signé est plus grand ou égal à **Rr** non signé. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si Rd >= Rr (C = 0) alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRSH	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	K000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si >= (C = 0)
        cp      r11, r12        ; Compare r11 et r12
        brsh    higtam          ; Branche si r11 >= r12 (non signé)
        ...
higtam: nop                     ; Branchement de destination
```

BRTC – Branch if the T Flag is Cleared – Branchement si Test T = 0

Branchement conditionnel relatif si test **T** est à 0. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si T = 0 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRTC	K	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	01kk	kkkk	k110

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si T = 0
        cp      r11, r12        ; Compare r11 et r12
        brtc    test            ; Branche si T = 0
        ...
test:   nop                     ; Branchement de destination
```

BRTS – Branch if the T Flag is Set – Branchement si Test T = 1

Branchement conditionnel relatif si test **T** est à 1. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si T = 1 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRTS	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	k110

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si T = 1
        cp      r11, r12        ; Compare r11 et r12
        brts    testset         ; Branche si T = 1
        ...
testset: nop                    ; Branchement de destination
```

BRVC – Branch if Overflow Cleared – Branchement si non Dépassement V = 0

Branchement conditionnel relatif si non dépassement **V** est à 0. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si V = 0 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRVC	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	01kk	kkkk	k011

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si V = 0
                cp      r11, r12        ; Compare r11 et r12
                brvc    noover          ; Branche si V = 0
                ...
noover:         nop                     ; Branchement de destination
```

BRVS – Branch if Overflow Set – Branchement si Dépassement V = 1

Branchement conditionnel relatif si dépassement **V** est à 1. Le branchement est relatif au **PC** dans l'une ou l'autre direction (**PC** -63 à **PC** +64). Le paramètre **k** est la compensation du **PC** représenté en complément à 2.

Si V = 1 alors PC = PC + k + 1, sinon PC = PC + 1

Avec : k = -64 à +63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BRVS	k	-	PC + k + 1 ou PC + 1	2	2 1

Code :

Code Instruction	1111	00kk	kkkk	k011

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Branchement si V = 1
                add     r11, r12        ; Addition de r11 et r12
                brvs    over            ; Branche si V = 1
                ...
over:           nop                     ; Branchement de destination
```

BSET – Bit Set in SREG – Mise à 1 du Bit ‘s’ de SREG

Active la valeur de la position du bit **s** correspondant dans le registre de statut **SREG**, la valeur de **s** est donc la position du bit dans le sens poids fort – poids faible. Les autres bits ne sont pas modifiés.

SREG(s) = 1

Avec : s = 0 à 7.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BSET	s	SREG	PC + 1	2	1

Code :

Code Instruction	1001	0100	0sss	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	X	X	X	X	X	X	X	X

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

Exemple :

```
; Active le bit s dans SREG
        bset    0           ; Active la retenue
        bset    7           ; Active les interruptions
```

BST – Bit Store fromBit in Register to T Flag in SREG – Stock Registre dans T

Stock le bit b de Rd dans le registre T de SREG :

T = Rd(b)

Avec : b = 0 à 7 et **Rd** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
BST	Rd, b	T	PC + 1	2	1

Code :

Code Instruction	1111	1010d	dddd	0bbb

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	X	-	-	-	-	-	-

T = Rd(b)

Exemple :

```
; Copie le bit Rd(b) dans T
        bst     r1, 2       ; Stock le bit 2 de r1 dans T
        bld     r0, 4       ; Charge T avec le bit 4 de r0
```

CALL – Long Call to a Subroutine – Appel Long de Sous-programme

Appel d'un sous-programme dans la mémoire programme entière. L'adresse de retour (l'instruction après **CALL**) sera stockée sur la pile et l'instruction **RET** rechargera cette dernière pour continuer le programme. :

PC = k avec modèle à PC de 16 bits soit 128Ko de mémoire programme maximum.

PC = k avec modèle à PC de 22 bits soit 8Mo de mémoire programme maximum.

Avec : k = 0 à 65 535 (16 bits) ou **k** = 0 à 8 388 607 (22 bits).

Instruction :

Code	Opérateurs	Pointeur Pile	Pile	PC	Octet	Cycle
CALL	k à 16 bits k à 22 bits	PS = PS – 2 PS = PS – 3	PC + 2	k	2 3	4 5

Code :

Code Instruction	1001	010k	kkkk	111k
Sur deux mots	kkkk	kkkk	kkkk	kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Appel de sous-programme
            mov     r16, r0         ; Copie r0 dans r16
            call    check           ; Appel sous-programme
            nop                     ; Continue ...
            ...
check:      cpi     r16, $42        ; Test si r16 est une valeur spéciale
            breq    error           ; Branche si égale
            ret                     ; Retour du sous-programme
            ...
error:      rjmp    error           ; Boucle infini (à ne pas faire)
```

CBI – Clear Bit in I/O Register – Mise à 0 d'un Registre d'Entrée/Sortie

Mise à 0 d'un bit indiqué dans un registre d'Entrée/Sortie. Cette instruction fonctionne sur les 32 registres inférieurs d'Entrée/Sortie d'adresse 0 à 31.

I/O(A, b) = 0

Avec : A = 0 à 31 et **b** = 0 à 7.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CBI	A, b	-	PC + 1	2	1

Code :

Code Instruction	1001	1000	AAAA	Abbb

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Mise à 0 du bit I/O(A,b)
            cbi     $12, 7          ; Efface le bit 7 du Port D
```

CBR – Clear Bits in Register – Efface les Bits d'un Registre en Complément à 1

Mise à 0 des bits indiqués dans le registre **Rd**. Exécute un ET logique entre le contenu du registre **Rd** et le complément à 1 du masque k. Le résultat sera placé dans le registre **Rd**. C'est l'opération inverse de **ANDI**.

Rd = Rd & ($FF - k)

Avec : Rd = 16 à 31 et **k** = 0 à 255.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CBR	Rd, k	Rd	PC + 1	2	1

Code :

Code Instruction	0111	kkkk	dddd	kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	X	0	X	X	-

S = N ⊕ V
V = 0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0

Exemple :

```
; Effacement logique Immédiat
        cbr     r16, $F0        ; Efface les bits de poids fort de r16
        cbr     r18, 1          ; Efface le bit 0 de r18
```

CLC – Clear Carry Flag – Efface la Retenue C = 0

Effacement de la Retenue **C** = 0 :

C = 0

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CLC	-	C = 0	PC + 1	2	1

Code :

Code Instruction	1001	0100	1000	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	0

C = 0

Exemple :

```
; Effacement C = 0
        add     r0, r0          ; Addition de r0 à lui même
        clc                     ; Efface le drapeau C = 0
```

CLH – Clear Half Carry Flag – Efface la Demi Retenue H = 0

Effacement de la demi Retenue **H** = 0 :

H = 0

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CLH	-	H = 0	PC + 1	2	1

Code :

Code Instruction	1001	0100	1101	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	**0**	-	-	-	-	-

H = 0

Exemple :

```
; Effacement H = 0
        clh                     ; Efface le drapeau H = 0
```

CLI – Clear Global Interrupt Flag – Désactive les Interruptions I = 0

Désactive les interruptions avec le drapeau **I** = 0 dans **SREG**. Les interruptions seront immédiatement mises hors de service et les interruptions ne seront pas exécutées même si elle arrive simultanément avec l'instruction **CLI**.

I = 0

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CLI	-	I = 0	PC + 1	2	1

Code :

Code Instruction	1001	0100	1111	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	**0**	-	-	-	-	-	-	-

I = 0

Exemple :

```
; Effacement I = 0
        in      temp, SREG      ; Stock SREG
        cli                     ; Arrêt des interruptions.
        Sbi     EECR, EEMWE     ; Début d'écriture EEPROM
        Sbi     EECR, EEWE
        Out     SREG, temp      ; Restore SREG (I=1)
```

CLN – Clear Negative Flag – Efface Négative N = 0

Effacement du Négatif **N** = 0 :

N = 0

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CLN	-	N = 0	PC + 1	2	1

Code :

Code Instruction	1001	0100	1010	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	**0**	-	-

C = 0

Exemple :

```
; Effacement N = 0
        cln                 ; Efface le drapeau N = 0
```

CLR – Clear Register – Effacement du Registre

Mise à 0 du registre. Cette instruction exécute un **OU Exclusif** entre un registre et lui-même. Cela mettra tous les bits à 0 dans le registre.

Rd = Rd Å **Rd**

Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CLR	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	0010	01dd	dddd	dddd

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	**0**	**0**	**0**	**1**	-

S = 0
V = 0
N = 0
Z = 1

Exemple :

```
; Effacement d'un registre
        clr     r18             ; Effacement de r18
loop:   inc     r18             ; Incrément de r18
        ...
        cpi     r18, $50        ; Compare r18 avec $50
        brne    loop
```

CLS – Clear Signed Flag – Efface le Signe S = 0

Effacement du Signe **S** = 0 :

S = 0

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CLS	-	S = 0	PC + 1	2	1

Code :

Code Instruction	1001	0100	1100	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	**0**	-	-	-	-

S = 0

Exemple :

```
cls                    ; Efface le drapeau S = 0
```

CLT – Clear T Flag – Efface le Test T = 0

Effacement du Test **T** = 0 :

T = 0

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CLT	-	T = 0	PC + 1	2	1

Code :

Code Instruction	1001	0100	1110	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	**0**	-	-	-	-	-	-

T = 0

Exemple :

```
clt                    ; Efface le drapeau T = 0
```

CLV – Clear Overflow Flag – Efface le Débordement V = 0

Effacement du Débordement **V** = 0 :

V = 0

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CLV	-	V = 0	PC + 1	2	1

Code :

Code Instruction	1001	0100	1011	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	**0**	-	-	-

V = 0

Exemple :

```
clv                    ; Efface le drapeau V = 0
```

CLZ – Clear Zero Flag – Efface le Zéro Z = 0

Effacement du Débordement **Z** = 0 :

Z = 0

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CLZ	-	Z = 0	PC + 1	2	1

Code :

Code Instruction	1001	0100	1001	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	**0**	-

Z = 0

Exemple :

```
clz                    ; Efface le drapeau Z = 0
```

COM – One's Complement – Complément à 1

Cette instruction effectue le complément à 1 de l'opérateur, c'est en fait l'inverse de la valeur :

Rd = $FF - Rd

Si vous avez **Rd** = $40, **COM Rd** sera égale à $BF, si l'on additionne ces deux valeur on retrouve $FF (256) ou encore en binaire :

$FF = b11111111
$ 40 = b00110000

$B0 = b11001111

Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
COM	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	0010	010d	dddd	0000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	**X**	**0**	**X**	**X**	**1**

S = N ⊕ V
V = 0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = 1

Exemple :

```
; Complément à 1
        com     r4              ; Complément à 1 de r4
        breq    zero            ; Branche si zéro
        ...
zero:   nop                     ; Branchement de destination
```

CP – Compare – Comparaison de Registre

Cette instruction exécute une comparaison entre deux registres **Rd** et **Rr**. Aucun des registres n'est changé. Tous les branchements conditionnels peuvent être employés après cette instruction.

Rd - Rr

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CP	Rd, Rr	-	PC + 1	2	1

Code :

Code Instruction	0001	01rd	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = Rd3 & Rr3 ! Rr3 & R3 ! R3 & Rd3
S = N ⊕ V
V = Rd7•Rr7•R7+Rd7•Rr7•R7
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = Rd7 & Rr7 ! Rr7 & R7 ! R7 & Rd7

Exemple :

```
        cp      r4, r19         ; Compare r4 avec r19
        brne    noteq           ; Branche si r4 <> r19
        ...
noteq:  nop                     ; Branchement de destination
```

CPC – Compare with Carry – Comparaison avec Retenue 16 bits

Cette instruction exécute une comparaison entre deux registres **Rd** et **Rr** et soustrait la retenue au résultat. Tous les branchements conditionnels peuvent être employés après cette instruction.

Rd – Rr - C

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CP	Rd, Rr	-	PC + 1	2	1

Code :

Code Instruction	0000	01rd	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = Rd3 & Rr3 ! Rr3 & R3 ! R3 & Rd3
S = N ⊕ V
V = Rd7•Rr7•R7+Rd7•Rr7•R7
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0 & Z
C = Rd7 & Rr7 ! Rr7 & R7 ! R7 & Rd7

Exemple :

```
; Comparaison r3:r2 avec r1:r0 sur 16 bits
        cp      r2, r0          ; Compare octet bas
```

```
                cpc     r3, r1          ; Compare octet haut avec Retenue
                brne    noteq           ; Branche si non égale
                ...
noteq:          nop                     ; Branchement de destination
```

CPI – Compare with Immediate – Comparaison avec Valeur Immédiate

Cette instruction exécute une comparaison entre un registre **Rd** et une constante **k**. Aucun des registres n'est changé. Tous les branchements conditionnels peuvent être employés après cette instruction.

Rd – k

Avec : Rd = 16 à 31 et **k** = 0 à 255.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CPI	Rd, k	-	PC + 1	2	1

Code :

Code Instruction	0011	kkkk	dddd	kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = Rd3 & Rr3 ! Rr3 & R3 ! R3 & Rd3
S = N ⊕ V
V = Rd7•Rr7•R7+Rd7•Rr7•R7
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = Rd7 & Rr7 ! Rr7 & R7 ! R7 & Rd7

Exemple :

```
                cpi     r16, $F0        ; Comparaison de r16 à $F0
                brne    error           ; Branche si r16 <> $F0
                ...
error:          nop                     ; Branchement de destination
```

CPSE – Compare Skip if Equal – Comparaison et Saut si Egale

Comparaison entre **Rd** et **Rr** et saute l'instruction suivante si égalité sinon on passe à l'instruction suivante :

Si Rd = Rr alors PC . PC + 2 (ou 3) sinon PC . PC + 1

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
CPSE	Rd, Rr	-	PC + 1 si faut	2	1
			PC + 2 si vrai	2	2
			PC + 3 (1)	2	3

Note 1 : Si le résultat est vrais et que l'instruction suivante à 3 octets, l'opération de saut sautera à PC + 3.

Code :

Code Instruction	0001	00rd	dddd	Rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
                inc     r4              ; Incrément r4
```

```
cpse    r4, r0          ; Compare r4 avec r0
neg     r4              ; Seulement si r4 <> r0
nop                     ; On saute ici si r4 = r0
```

DEC – Decrement - Décrémentation

Soustrait 1 du contenu du registre **Rd** et place le résultat dans **Rd**. Le drapeau **C** dans **SREG** n'est pas affecté par l'opération, permettant ainsi à l'instruction **DEC** d'être employé comme un compteur de boucle dans des calculs de précision multiple. On peut réaliser des boucles d'attente en faisant fonctionner **DEC** sur des valeurs non signées avec **BREQ** et **BRNE**. En utilisant des valeurs en complément à 2, tous les branchements signés sont disponibles.

Rd = Rd - 1

Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
DEC	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	1001	010d	dddd	1010

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	X	X	X	X	-

S = N ⊕ V
V = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0

Exemple :

```
        ldi     r17, $10        ; Charge $10 dans r17
loop:   add     r1, r2          ; Addition de r2 à r1
        dec     r17             ; Décrément de r17
        brne    loop            ; Branche si r17 <> 0
        nop                     ; Continue
```

EICALL – Extended Indirect Call to Subroutine – Appel Indirect au Sous-Programme

Appel indirect d'un sous-programme indiqué par le registre **Z** et le registre **EIND** dans l'espace d'Entrée/Sortie. Cette instruction tient compte d'appel indirect sur l'espace mémoire programme entier. L'indicateur de pile est décrémenté pendant **EICALL**. Cette instruction n'est pas mise en oeuvre pour les modèles avec un **PC** sur 2 octets, voir **ICALL**.

PC(15:0) = Z et PC(21:16) = EIND (modèle à PC de 22 bits)

Avec : Z = 0 à 65 535 et **EIND** = 0 à 256 (22 bits au total).

Instruction :

Code	Opérateurs	Pointeur Pile	Pile	PC	Octet	Cycle
EICALL	Z, EIND	PS = PS – 3	PC + 1	Z + EIND	2	4

Code :

Code Instruction	1001	0101	0001	1001

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Appel de sous-programme
        ldi     r16, $05        ; Charge EIND et Z
        out     EIND, r16       ; Adresse de poids fort
        ldi     r30, $00        ; Adresse de poids faible
        ldi     r31, $10        ; Adresse de poids moyen
        eicall                  ; Call à l'adresse $051000
```

EIJMP – Extended Indirect Jump – Saut Indirect avec Z & EIND

Saut indirect à l'adresse indiquée par **Z** et le registre **EIND** dans l'espace d'Entrée/Sortie. Cette instruction tient compte des sauts indirects dans l'espace mémoire programme entier. Cette instruction n'est pas disponible dans tous les modèles.

PC(15:0) = Z et PC(21:16) = EIND (modèle à PC de 22 bits)

Avec : Z = 0 à 65 535 et **EIND** = 0 à 256 (22 bits au total).

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
EIJMP	Z, EIND	-	Z + EIND	2	2

Code :

Code Instruction	1001	0100	0001	1001

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Saut Indirect
        ldi     r16, $05        ; Charge EIND et Z
        out     EIND, r16       ; Adresse de poids fort
        ldi     r30, $00        ; Adresse de poids faible
        ldi     r31, $10        ; Adresse de poids moyen
        eijmp                   ; Saute à l'adresse $051000
```

ELPM – Extended Load Program Memory – Chargement Programme Mémoire Etendue

Charge un octet indiqué par le registre **Z** et le registre **RAMPZ** dans l'espace d'Entrée/Sortie et place cet octet dans **Rd**. La mémoire programme est organisée en mot de 16 bits tandis que Z pointe une adresse 8 d'un octet. Ainsi, le bit le moins significatif de Z choisit l'octet bas (**ZLSB** = 0) ou l'octet haut (**ZLSB** = 1). Cette instruction peut adresser l'espace mémoire programme entier. Le registre Z peut être laissé inchangé par l'opération, ou il peut être incrémenté. L'augmentation s'applique à la concaténation entière sur 24 bits de **Z** et **RAMPZ**. La programmation système peut employer l'instruction **ELPM** pour lire les fusibles et la valeur des bits de blocage. Référez-vous à la documentation Anglaise pour une description détaillée.

Le résultat de ces combinaisons est non défini (ne pas utiliser) : **ELPM** r30, **Z**+ et **ELPM** r31, **Z**+.

Rd = RAMPZ:Z

Avec : Rd = 0 à 29 et **Z** = 0 à 65 535 et **RAMPZ** = 0 à 256.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ELPM	Z, RAMPZ	Rd	PC + 1	2	3

Code :

Code Instruction	ELPM (r0 implicite)	1001	0101	1101	1001
Suite	ELPM Z (ou Z+)	1001	010d	dddd	0111

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Programmation Mémoire FLASH
        ldi     ZL, byte3(Table_1<<1)   ; Initialise Z avec la table octet 3
        out     RAMPZ, ZL               ; Sauve la RampZ
        ldi     ZH, byte2(Table_1<<1)   ; Charge Z avec octet 2
        ldi     ZL, byte1(Table_1<<1)   ; Charge Z avec octet 1
        elpm    r16, Z+                 ; Charge la constante du Programme
        ...
Table_1:                                ; Mémoire pointé par RAMPZ:Z
        dw      $3738                   ; $38 est l'adressé quant ZLSB = 0
                                        ; $37 est l'adressé quant ZLSB = 1
```

EOR – Exclusive OR – OU exclusif

Exécute un OU logique exclusif entre le contenu **Rd** et **Rr** et place le résultat dans **Rd** :

Rd = Rd Å Rr

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
EOR	Rd, Rr	Rd	PC + 1	2	1

Code :

Code Instruction	0010	01rd	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	X	0	X	X	-

S = N ⊕ V
V = 0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0

Exemple :

```
; OU logique exclusif
        eor     r4, r4          ; Efface r4
        eor     r0, r22         ; OU Exclusif sur les Bits r0 et r22
```

FMUL – Fractional Multiply Unsigned – Multiplication Fractionnaire non Signé

Cette instruction exécute une multiplication avec deux registres à 8 bits et un résultat sur 16 bits non signé.

MUL est plus généralement employé avec des nombres signés, tandis que **FMUL** est utilisé avec des nombres non signés. Cette instruction est donc la plus utile pour le calcul d'un produit partiel en exécutant une multiplication signée avec des entrées 16 bits. Le résultat de l'opération **FMUL** peut souffrir d'un complément à 2 sur débordement.

Le multiplicande **Rd** et le multiplicateur **Rr** sont deux registres contenant les nombres non signés fractionnaires. Le produit fractionnaire à 16 bits non signé est placé dans **R1** (poids fort) et **R0** (poids faible). Cette instruction n'est pas disponible dans tous les modèles.

R1:R0 (non signé) = Rd (non signé) x Rr (non signé)

Avec : Rd = 16 à 23 et **Rr** = 16 à 23.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
FMUL	Rd, Rr	R1:R0	PC + 1	2	2

Code :

Code Instruction	0000	0011	0ddd	1rrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	X	X

Z = R15&R14&R13&R12&R11&R10 & R9 & R8 & R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = R16 (R15 -> C)

Exemple :

```
; Multiplication deux fois 16 bits, résultat sur 32 bits signés
fmuls16x16_32:                  ; r19:r18:r17:r16 = ( r23:r22 * r21:r20 )
        clrr2
        fmuls   r23, r21        ; ((signé)ah * (signé)bh) << 1
```

```
movw  r19:r18, r1:r0
fmul  r22, r20          ; (al * bl) << 1
adc   r18, r2
movw  r17:r16, r1:r0
fmulsu r23, r20         ; ((signé)ah * bl) << 1
sbc   r19, r2
add   r17, r0
adc   r18, r1
adc   r19, r2
fmulsu r21, r22         ; ((signé)bh * al) << 1
sbc   r19, r2
add   r17, r0
adc   r18, r1
adc   r19, r2
```

FMULS – Fractional Multiply Signed – Multiplication Fractionnaire Signé

Cette instruction exécute une multiplication avec deux registres à 8 bits et un résultat sur 16 bits signé.

Le multiplicande **Rd** et le multiplicateur **Rr** sont deux registres contenant les nombres signés fractionnaires. Le produit fractionnaire à 16 bits signé est placé dans **R1** (poids fort) et **R0** (poids faible). Cette instruction n'est pas disponible dans tous les modèles.

R1:R0 (singé)= Rd (singé) x Rr (singé)

Avec : Rd = 16 à 23 et **Rr** = 16 à 23.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
FMULS	Rd, Rr	R1:R0	PC + 1	2	2

Code :

Code Instruction	0000	0011	1ddd	0rrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	X	X

Z = R15&R14&R13&R12&R11&R10 & R9 & R8 & R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = R16 (R15 -> C)

Exemple :

```
; Multiplication deux fois 8 bits, résultat sur 16 bits signés
        fmuls   r23, r22        ; Multiplication signé entre r23 et r22
        movw    r23:r22, r1:r0  ; Copie le résultat dans r23:r22
```

FMULSU – Fractional Multiply Signed with Unsigned – Multipication Signé-non Signé

Cette instruction exécute une multiplication avec deux registres à 8 bits signé et non signé et un résultat sur 16 bits signé.

Le multiplicande **Rd** contient un nombre signé fractionnaire et le multiplicateur **Rr** contient un nombre non signé fractionnaire. Le produit fractionnaire à 16 bits signé est placé dans **R1** (poids fort) et **R0** (poids faible). Cette instruction n'est pas disponible dans tous les modèles.

R1:R0 (singé) = Rd (signé) x Rr (non signé)

Avec : Rd = 16 à 23 et **Rr** = 16 à 23.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
FMULSU	Rd, Rr	R1:R0	PC + 1	2	2

Code :

Code Instruction	0000	0011	1ddd	1rrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	X	X

Z = R15&R14&R13&R12&R11&R10 & R9 & R8 & R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = R16 (R15 -> C)

Exemple :

```
; Multiplication deux fois 8 bits, résultat sur 16 bits signés
fmuls16x16_32:                  ; r19:r18:r17:r16 = ( r23:r22 * r21:r20 )
        clrr2
        fmuls   r23, r21        ; ((signé)ah * (signé)bh) << 1
        movw    r19:r18, r1:r0
        fmul    r22, r20        ; (al * bl) << 1
        adc     r18, r2
        movw    r17:r16, r1:r0
        fmulsu  r23, r20        ; ((signé)ah * bl) << 1
        sbc     r19, r2
        add     r17, r0
        adc     r18, r1
        adc     r19, r2
        fmulsu  r21, r22        ; ((signé)bh * al) << 1
        sbc     r19, r2
        add     r17, r0
        adc     r18, r1
        adc     r19, r2
```

ICALL – Indirect Call to Subroutine – Appel de Sous-programme Indirect

Appel indirect d'un sous-programme indiqué par le registre **Z** sur une plage large de 64 Ko dans l'espace de mémoire programme. L'indicateur de pile est décrémenté pendant **ICALL**.

PC = Z avec modèle à PC de 16 bits soit 128Ko de mémoire programme maximum.

PC(15:0) = Z, PC(21:16)=0, avec modèle à PC de 22 bits.

Avec : Z = 0 à 65 535 (16 bits ou 22 bits).

Instruction :

Code	Opérateurs	Pointeur Pile	Pile	PC	Octet	Cycle
ICALL	Z	PS = PS – 2 PS = PS – 3	PC + 1	Z	2	3 4

Code :

Code Instruction	1001	0101	0000	1001

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Appel de sous-programme indirect
        mov     r30, r0         ; Copie r0 dans r30
        icall   check           ; Appel sous-programme pointé par r31:r30
        nop                     ; Continue ...
        ...
```

IJMP – Indirect Jump – Saut Indirect

Saut indirect à l'adresse indiquée par **Z** sur une plage large de 64 Ko dans l'espace mémoire programme. Cette instruction n'est pas disponible dans tous les modèles.

PC = Z avec modèle à PC de 16 bits soit 128Ko de mémoire programme maximum.

PC(15:0) = Z, PC(21:16)=0, avec modèle à PC de 22 bits.

Avec : Z = 0 à 65 535 (16 bits ou 22 bits).

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
IJMP	Z	-	Z	2	2

Code :

Code Instruction	1001	0100	0000	1001

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Saut Indirect
        mov     r30, r0         ; Copie de r0 dans Z (r30)
        ijmp                    ; Saut au programme pointé par r31:r30
```

IN - Load an I/O Location to Register – Lecture d'une Entrée/Sortie

Lecture d'une donnée de l'espace d'Entrée/Sortie (Ports, Timer, Configuration, …) et la placer dans le registre **Rd :**

Rd = I/O(A)

Avec : Rd = 0 à 31 et **A** = 0 à 63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
IN	Rd, A	Rd	PC + 1	2	1

Code :

Code Instruction	1011	0AAd	dddd	AAAA

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Lecture d'un port
        in      r25, $16        ; Lecture Port B
        cpi     r25, 4          ; Compare avec une constante 4
        breq    exit            ; Branche si r25=4
        ...
exit:   nop                     ; Branchement destination
```

INC – Increment - Incrément

Ajoute 1 au contenu du registre **Rd** et place le résultat dans **Rd**. Le drapeau **C** dans **SREG** n'est pas affecté par l'opération, permettant ainsi à l'instruction **INC** d'être employé comme un compteur de boucle dans des calculs de précision multiple. On peut réaliser des boucles d'attente en faisant fonctionner **INC** sur des valeurs non signées avec **BREQ** et **BRNE**. En utilisant des valeurs en complément à 2, tous les branchements signées sont disponibles.

Rd = Rd + 1

Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
INC	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	1001	010d	dddd	0011

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	**X**	**X**	**X**	**X**	-

S = N ⊕ V
V = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0

Exemple :

```
; Incrémentation
        ldi     r17, $10        ; Charge $10 dans r17
loop:   add     r1, r2          ; Addition de r2 à r1
        inc     r17             ; Incrément de r17
        cpi     r17, $4F        ; Compare r22 à $4f
```

```
        brne    loop            ; Branche si r17 <> $4f
        nop                     ; Continue
```

JMP – Jump - Saut

Saut à l'adresse indiquée par **k** sur une plage large de 4 Mo dans l'espace mémoire programme. Cette instruction n'est pas disponible dans tous les modèles.

PC = k

Avec : k = 0 à 4 194 303 (22 bits).

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
JMP	k	-	k	4	3

Code :

Code Instruction	1001	010k	kkkk	110k
Suite du code	kkkk	kkkk	kkkk	kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Saut inconditionnel
        mov     r1,r0           ; Copie r0 dans r1
        jmp     farplc          ; Saut inconditionnel
        ...
farplc: nop                     ; Destination du saut
```

LD – Load Indirect from Data Space to Register using Index X – Charge Indirect X

Charge un octet indirectement via **X** de l'espace de données sur un registre d'accès rapide. L'espace de données est soit les registres d'accès rapide, soit la mémoire d'Entrée/Sortie, soit la **SRAM** interne. **L'EEPROM** a un espace d'adresse séparé non accessible.

L'emplacement de la données est indiqué par le registre **X** (16 bits). L'accès de la mémoire est limité au segment de données sur 64Ko. Pour avoir accès à un autre segment de données dans les modèles avec plus d'espace, la **RAMPX** dans le secteur d'Entrée/Sortie doit être changé.

Le Registre **X** peut être laissé inchangé par l'opération, ou il peut être post-incrémenté ou pré-décrémenté. Ces particularités sont utiles pour l'accès aux tableaux, tables et empilement, avec une restriction à seulement 256 octets, car seul l'octet bas de **X** est mis à jour. L'octet haut de **X** n'est pas employé par cette instruction et peut être employé pour d'autres buts. Attention, le résultat de ces combinaisons utilisant le registre **X** est non défini : **LD r26, X+ ; LD r27, X+ ; LD r26, -X ; LD r27, -X**.

Rd = (X) [1] ou Rd = (X), X = X + 1 [2], ou X = X – 1, Rd = (X) [3]

Avec : Rd = 0 à 31 (sauf 26 et 27).

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
LD	(X)	Rd	PC + 1	2	2

Code :

Code Instruction 1	1001	010d	dddd	1100
Code Instruction 2	1001	010d	dddd	1101
Code Instruction 3	1001	010d	dddd	1110

En fonction de l'opération simple [1], de la post-incrémentation [2] ou de la pré-incrémentation [3], le code instruction change.

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Charge indirect X
        clr     r27             ; Efface X bits haut
        ldi     r26, $60        ; X bas à $60
        ld      r0, X+          ; Charge r0 avec la donnée à l'adresse $60(X post inc)
        ld      r1, X           ; Charge r1 avec la donnée à l'adresse $61
        ldi     r26, $63        ; X bas à $63
        ld      r2, X           ; Charge r2 avec la donnée à l'adresse $63
        ld      r3, –X          ; Charge r3 avec la donnée à l'adresse $62(X pre dec)
```

LD (LDD) – Load Indirect from Data Space to Register using Index Y – Charge Indirect Y

Charge un octet indirectement via **Y** de l'espace de données sur un registre d'accès rapide. L'espace de données est soit les registres d'accès rapide, soit la mémoire d'Entrée/Sortie, soit la **SRAM** interne. **L'EEPROM** a un espace d'adresse séparé non accessible.

L'emplacement de la données est indiqué par le registre **Y** (16 bits). L'accès de la mémoire est limité au segment de données sur 64Ko. Pour avoir accès à un autre segment de données dans les modèles avec plus d'espace, la **RAMPY** dans le secteur d'Entrée/Sortie doit être changé.

Le Registre **Y** peut être laissé inchangé par l'opération, ou il peut être post-incrémenté ou pré-décrémenté. Ces particularités sont utiles pour l'accès aux tableaux, tables et empilement, avec une restriction à seulement 256 octets, car seul l'octet bas de **Y** est mis à jour. L'octet haut de **Y** n'est pas employé par cette instruction et peut être employé pour d'autres buts. Attention, le résultat de ces combinaisons utilisant le registre **Y** est non défini : **LD r28, Y+ ; LD r29, Y+ ; LD r28, -Y ; LD r29, -Y**.

Rd = (Y) [1] ou Rd = (Y), Y = Y + 1 [2], ou Y = Y – 1, Rd = (Y) [3], ou Rd = (Y + q) [4]

Avec : Rd = 0 à 31 (sauf 28 et 29), **q** = 0 à 63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
LD & LDD	(Y), q	Rd	PC + 1	2	2

Code :

Code Instruction 1	1000	000d	dddd	1100
Code Instruction 2	1001	000d	dddd	1101
Code Instruction 3	1001	000d	dddd	1110
Code Instruction 4	10q0	qq0d	dddd	1qqq

En fonction de l'opération simple [1], de la post-incrémentation [2] ; de la pré-incrémentation [3] ou du décalage dans l'espace d'Entrée/Sortie [4], le code instruction change.

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Charge indirect Y
        clr     r29             ; Efface Y bit haut
        ldi     r28, $60        ; Y bas à $60
        ld      r0, Y+          ; Charge r0 avec la donnée à l'adresse $60(Y post inc)
        ld      r1, Y           ; Charge r1 avec la donnée à l'adresse $61
        ldi     r28, $63        ; Y bas à $63
        ld      r2, Y           ; Charge r2 avec la donnée à l'adresse $63
        ld      r3, -Y          ; Charge r3 avec la donnée à l'adresse $62(Y pre dec)
        ldd     r4, Y+2         ; Charge r4 avec la donnée à l'adresse $64
```

LD (LDD) – Load Indirect From Data Space to Register using Index Z – Charge Indirect Z

Charge un octet indirectement via **Z** de l'espace de données sur un registre d'accès rapide. L'espace de données est soit les registres d'accès rapide, soit la mémoire d'Entrée/Sortie, soit la **SRAM** interne. **L'EEPROM** a un espace d'adresse séparé non accessible.

L'emplacement de la données est indiqué par le registre **Z** (16 bits). L'accès de la mémoire est limité au segment de données sur 64Ko. Pour avoir accès à un autre segment de données dans les modèles avec plus d'espace, la **RAMPY** dans le secteur d'Entrée/Sortie doit être changé.

Le Registre **Z** peut être laissé inchangé par l'opération, ou il peut être post-incrémenté ou pré-décrémenté. Ces particularités sont utiles pour l'accès aux tableaux, tables et empilement, avec une restriction à seulement 256 octets, car seul l'octet bas de **Z** est mis à jour. L'octet haut de **Z** n'est pas employé par cette instruction et peut être employé pour d'autres buts. Attention, le résultat de ces combinaisons utilisant le registre **Z** est non défini : **LD r30, Z+ ; LD r31, Z+ ; LD r30, -Z ; LD r31, -Z**.

Rd = (Z) [1] ou Rd = (Z), Z = Z + 1 [2], ou Z = Z – 1, Rd = (Z) [3], Rd = (Z + q) [4]

Avec : Rd = 0 à 29 (sauf 30 et 31), **q** = 0 à 63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
LD & LDD	(Z), q	Rd	PC + 1	2	2

Code :

Code Instruction 1	1000	000d	dddd	0000
Code Instruction 2	1001	000d	dddd	0001
Code Instruction 3	1001	000d	dddd	0010
Code Instruction 4	10q0	qq0d	dddd	0qqq

En fonction de l'opération simple [1], de la post-incrémentation [2] ; de la pré-incrémentation [3] ou du décalage dans l'espace d'Entrée/Sortie [4], le code instruction change.

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Charge indirect Z
        clr     r31             ; Efface Z haut
        ldi     r30,$60         ; Z bas à $60
        ld      r0,Z+           ; Charge r0 avec la donnée à l'adresse $60(Z post inc)
        ld      r1,Z            ; Charge r1 avec la donnée à l'adresse $61
        ldi     r30,$63         ; Z bas à $63
        ld      r2,Z            ; Charge r2 avec la donnée à l'adresse $63
        ld      r3,-Z           ; Charge r3 avec la donnée à l'adresse $62(Z pre dec)
        ldd     r4,Z+2          ; Charge r4 avec la donnée à l'adresse $64
```

LDI – Load Immediate – Charge Valeur Immédiate

Charge la valeur immédiate d'un octet sur un registre d'accès rapide.

Rd = K

Avec : Rd = 16 à 31, **K** = 0 à 255.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
LDI	Rd, K	Rd	PC + 1	2	1

Code :

Code Instruction	1110	KKKK	dddd	KKKK

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Charge immédiat
        clr     r31             ; Efface Z haut
        ldi     r30, $F0        ; Z bas à $F0
        lpm                     ; Charge les constantes du programme
; Mémoire pointée par Z
```

LDS – Load Direct from Data Space – Charge Direct

Charge un octet directement de l'espace de données sur un registre d'accès rapide. L'espace de données est soit les registres d'accès rapide, soit la mémoire d'Entrée/Sortie, soit la **SRAM** interne. **L'EEPROM** a un espace d'adresse séparé non accessible.

Une adresse 16 bits doit être fournie. L'accès de la mémoire est limité au segment de données de 64Ko. L'instruction **LDS** emploie le registre de **RAMPD** pour avoir accès à la mémoire supérieur à 64Ko.

Rd = (k)

Avec : Rd = 16 à 31, **k** = 0 à 65535.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
LDS	Rd, k	Rd	PC + 1	4	2

Code :

Code Instruction	1001	000d	dddd	0000
Suite instruction	kkkk	kkkk	kkkk	Kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Charge direct mémoire
        lds     r2, $FF00       ; Charge r2 avec le contenu de l'adresse $FF00
        add     r2, r1          ; Ajout r1 à r2
        sts     $FF00, r2       ; Ecrire en retour
```

LPM – Load Program Memory – Charge un Programme en Mémoire

Charge un octet indiqué par le registre **Z** dans la destination **Rd**. La mémoire programme est organisée en mots de 16 bits alors que **Z** est une adresse en octet. Ainsi, le bit le moins significatif de **Z** choisit l'octet bas ($ZLS_B = 0$) ou l'octet haut ($ZLS_B = 1$). Cette instruction adresse les 64Ko (par mots de 32Ko) de mémoire programme. Le registre **Z** peut être laissé inchangé par l'opération, ou il peut être incrémenté. L'augmentation ne s'applique pas au registre de **RAMPZ**.

Les modèles avec programmation interne peuvent employer l'instruction **LPM** pour lire les fusibles et les valeurs des bits de blocages. Référez-vous à la documentation sur la programmation du microcontrôleur.

Le résultat de ces combinaisons est non défini : **LPM r30, Z+, LPM r31, Z+.**

R0 = (Z) [1] ou Rd = (Z) [2], ou Rd = (Z), Z = Z + 1 [3]

Avec : Rd = 0 à 29 (sauf 30 et 31).

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
LPM	(Z)	Rd	PC + 1	2	3

Code :

Code Instruction 1	1001	1010	1100	1000
Code Instruction 2	1001	000d	dddd	0100
Code Instruction 3	1001	000d	dddd	0101

En fonction de l'opération directe sur R0 [1], avec tous les registres d'accès rapide [2], ou de la post-incrémentation [3], le code instruction change.

Registre Statut :

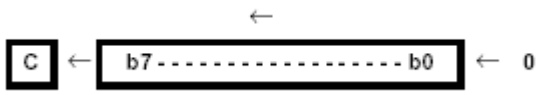
SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Charge programme en mémoire
        ldi     ZH, high(Table_1<<1)    ; Initialise Z avec l'adresse de la table
        ldi     ZL, low(Table_1<<1)
        lpm     r16, Z                  ; Charge la constante du Programme
; Mémoire pointée par Z (r31:r30)
        ...
Table_1:                                ; Adresse de la zone à programmer
        Dw      $5876                   ; $76 est l'adresse avec ZLSB = 0
                                        ; $58 est l'adresse avec ZLSB = 1
```

LSL – Logical Shift Left – Décalage Logique à Gauche

Décalage de tous les bits dans **Rd** d'une place à gauche, le bit 0 est effacé, le bit 7 est placé dans **C**. Cette opération multiplie efficacement les valeurs signées et non signées par deux.



Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
LSL	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	0000	11dd	dddd	Dddd

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

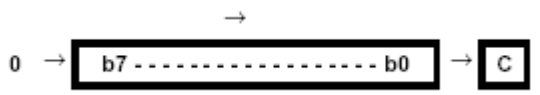
H = Rd3
S = N ⊕ V
V = N ⊕ C
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = Rd7

Exemple :

```
add     r0, r4          ; Ajout de r4 à r0
lsl     r0              ; Multiplie r0 par 2
```

LSR – Logical Shift Right – Décalage Logique à Droite

Décalage de tous les bits dans **Rd** d'une place à droite, le bit 7 est effacé, le bit 0 est placé dans **C**. Cette opération divise efficacement les valeurs non signées par deux.



Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
LSR	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	1001	010d	dddd	0110

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	X	X	0	X	X

S = N ⊕ V
V = N ⊕ C
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = Rd0

Exemple :

```
add     r0, r4          ; Addition r4 à r0
lsr     r0              ; Divise r0 par 2
```

MOV – Copy Register – Copie de Registre

Cette instruction fait une copie d'un registre dans un autre. Le registre source **Rr** est laissé inchangé, tandis que la destination se fait dans **Rd**.

Rd = Rr

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
MOV	Rd, Rr	Rd	PC + 1	2	1

Code :

Code Instruction	0010	11rd	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Copie de registre
        mov   r16, r0          ; Copie r0 dans r16
        call  check            ; Appel sous-programme
        ...
check:  cpi   r16, $11         ; Compare r16 et $11
        ...
        ret                    ; Retour de sous-programme
```

MOVW– Copy Register Word – Copie d'un Registre sur 16 bits

Cette instruction fait une copie d'une paire de registre dans un autre. Le registre source **Rr1:0** est laissé inchangé, tandis que la destination se fait dans **Rd1:0**.

Rd1:0 = Rr1:0

Avec : Rd = 0 à 30 de 2 en 2 et **Rr** = 0 à 30 de 2 en 2.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
MOVW	Rd, Rr	Rd	PC + 1	2	1

Code :

Code Instruction	0000	0001	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Copie de Registre 16 bits
        movw  r17:16, r1:r0    ; Copie r1:r0 avec r17:r16
        call  check            ; Appel sous-programme
        ...
check:  cpi   r16, $11         ; Compare r16 avec $11
        ...
        cpi   r17, $32         ; Compare r17 avec $32
        ...
        ret                    ; Retour de sous-programme
```

MUL – Multiply Unsigned – Multiplication non Signé

L'instruction multiplie deux registres 8 bits, le résultat est placé dans le registre à 16 bits **R1** et **R0**.

Le multipliant **Rd** et le multiplicateur **Rr** sont deux registres contenant des nombres non signés. Le produit 16 bits non signé est placé dans **R1** (l'octet haut) et **R0** (l'octet bas). Si le multipliant ou le multiplicateur sont **R0** ou **R1** le résultat recopiera ceux-là après la multiplication.

R1:R0 = Rd x Rr

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
MUL	Rd, Rr	R1:R0	PC + 1	2	2

Code :

Code Instruction	1001	11rd	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	X	X

Z = R15&R14&R13&R12&R11&R10&R9&R8&R7&R6& R5 & R4 & R3 & R2 & R1 & R0

C = R15

Exemple :

```
; Multiplication 2 x 8 bits = 16 bits
        mul     r5, r4          ; Multiplie non signé r5 et r4
        movw    r4, r0          ; Copie le résultat après dans r5:r4
```

MULS – Multiply Signed – Multiplication Signé

L'instruction multiplie deux registres 8 bits signés, le résultat signé est placé dans le registre à 16 bits **R1** et **R0**.

Le multipliant **Rd** et le multiplicateur **Rr** sont deux registres contenant des nombres signés. Le produit 16 bits signé est placé dans **R1** (l'octet haut) et **R0** (l'octet bas).

R1:R0 (signé) = Rd (signé) x Rr (signé)

Avec : Rd = 16 à 31 et **Rr** = 16 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
MULS	Rd, Rr	R1:R0	PC + 1	2	2

Code :

Code Instruction	0000	0010	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	X	X

Z = R15&R14&R13&R12&R11&R10&R9&R8&R7&R6& R5 & R4 & R3 & R2 & R1 & R0

C = R15

Exemple :

```
; Multiplication 2 x 8 bits = 16 bits
        muls    r21, r20        ; Multiplication signé r21 et r20
        movw    r20, r0         ; Copie la résultat dans r21:r20
```

MULSU – Multiply Signed with Unsigned – Multiplication Signé et non Signé

L'instruction multiplie deux registres 8 bits, un signé l'autre non signé, le résultat signé est placé dans le registre à 16 bits **R1** et **R0**.

Le multipliant **Rd** est un nombre signé et le multiplicateur **Rr** est un nombre non signé. Le produit 16 bits signé est placé dans **R1** (l'octet haut) et **R0** (l'octet bas).

R1:R0 (signé) = Rd (signé) x Rr (non signé)

Avec : Rd = 16 à 23 et **Rr** = 16 à 23.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
MULSU	Rd, Rr	R1:R0	PC + 1	2	2

Code :

Code Instruction	0000	0011	0ddd	0rrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	X	X

Z = R15&R14&R13&R12&R11&R10&R9&R8&R7&R6& R5 & R4 & R3 & R2 & R1 & R0

C = R15

Exemple :

```
; Multiplication 2 x 16 bits = 32 bits (r19:r18:r17:r16 = r23:r22 * r21:r20)
muls16x16_32:                       Multiplication 16 * 16
        clr     r2                  ; Efface r2
        muls    r23, r21            ; (signé) ah * (signé) bh
        movw    r19:r18, r1:r0      ; Copie le résultat
        mul     r22, r20            ; al * bl
        movw    r17:r16, r1:r0      ; Copie le résultat
        mulsu   r23, r20            ; (signed)ah * bl
        sbc     r19, r2
        add     r17, r0
        adc     r18, r1
        adc     r19, r2
        mulsu   r21, r22            ; (signé) bh * al
        sbc     r19, r2
        add     r17, r0
        adc     r18, r1
        adc     r19, r2
        ret                         ; Retour du sous-programme
```

NEG – Two's Complement – Complément à Deux

Remplace le contenu de registre **Rd** avec le complément à deux; le cas particulier de la valeur $80 est laissé inchangé :

Rd = $00 - Rd

Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
NEG	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	1001	010d	dddd	0001

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = R3 + Rd3
S = N ⊕ V
V = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0

Exemple :

```
; Complément à deux
        sub     r11, r0         ; Soustrait r0 de r11
        brpl    positive        ; Branche si résultat positif
        neg     r11             ; Complément à deux de r11
positive:   nop                 ; Destination du branchement
```

NOP – No Operation – Pas d'Opération

Cette instruction consomme 1 cycle d'horloge à ne rien faire (attente).

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
NOP	-	-	PC + 1	2	1

Code :

Code Instruction	0000	0000	0000	0000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Attente d'un cycle
        clr     r16             ; Efface r16
        ser     r17             ; Active r17
        out     $18, r16        ; Ecrire sur le port B
        nop                     ; Attendre la synchronisation
        out     $18, r17        ; Ecrire sur le port B
```

OR – Logical OR – OU Logique

Exécute un **OU** logique entre le contenu de **Rd** et **Rr** et place le résultat dans **Rd** :

Rd = Rd ! Rr

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
OR	Rd, Rr	Rd	PC + 1	2	1

Code :

Code Instruction	0010	10rd	Dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	X	0	X	X	-

S = N ⊕ V
V = 0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0

Exemple :

```
; Ou logique
        or      r15, r16        ; Test les deux registres
        bst     r15, 6          ; Stock le bit 6 de r15 dans T
        brts    ok              ; Branche si T est à 1
        ...
ok:     nop                     ; Destination du branchement
```

ORI – Logical OR with Immediate – OU Logique Immédiat

Exécute un **OU** logique entre le contenu de **Rd** et une valeur immédiate et place le résultat dans **Rd** :

Rd = Rd ! K

Avec : Rd = 16 à 31 et **K** = 0 à 255.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ORI	Rd, K	Rd	PC + 1	2	1

Code :

Code Instruction	0110	KKKK	dddd	KKKK

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	X	0	X	X	-

S = N ⊕ V
V = 0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0

Exemple :

```
; Ou logique
        ori     r16, $F0        ; Met les 4 bits de poids fort de r16 à 1
        ori     r17, 1          ; Met le bit 0 de r17 à 1
```

OUT – Store Register to I/O Location – Ecriture d'une Entrée/Sortie

Ecriture d'une donnée présente sur le registre **Rr** dans l'espace d'Entrée/Sortie (Ports, Timer, Configuration, …) **:**

I/O(A) = Rr

Avec : Rr = 0 à 31 et **A** = 0 à 63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
OUT	Rr, A	-	PC + 1	2	1

Code :

Code Instruction	1011	1AAr	rrrr	AAAA

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Ecriture d'un port
        clr     r16             ; Efface r16
        ser     r17             ; r17 = 1
        out     $18, r16        ; Ecrire 0 sur le Port B
        nop                     ; Attendre la synchronisation
        out     $18, r17        ; Ecrire 1 sur le Port B
```

POP – Pop Register from Stack – Restaure la Donnée de la Pile

Restauration de la donnée de la pile sur un registre d'accès rapide. L'instruction lie la valeur de la pile et le stock dans le registre **Rd**. L'indicateur de pile est pré incrémenté de 1 avant d'exécuter l'instruction **POP**.

Rd = PILE

Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	Pointeur Pile	PC	Octet	Cycle
POP	Rd	Rd	PS = PS + 1	PC + 1	2	2

Code :

Code Instruction	1001	000d	Dddd	1111

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Restaure de la pile sur un registre
        call    routine         ; Appel de sous-programme
        ...
routine: push   r14             ; Stock r14 sur la pile
        push    r13             ; Stock r13 sur la pile
        ...
        pop     r13             ; Restaure r13
        pop     r14             ; Restaure r14
        ret                     ; Retour de sous-programme
```

PUSH – Push Register on Stack – Stock un Registre sur la Pile

Stock la donnée d'un registre d'accès rapide sur la pile sur. L'instruction lie la valeur du registre **Rr** et le stock sur la pile. L'indicateur de pile est post décrémenté de 1 avant d'exécuter l'instruction **PUSH**.

PILE = Rr

Avec : Rr = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	Pointeur Pile	PC	Octet	Cycle
PUSH	Rr	-	PS = PS - 1	PC + 1	2	2

Code :

Code Instruction	1001	001r	rrrr	1111

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
            call    routine         ; Appel de sous-programme
            ...
routine:    push    r14             ; Stock r14 sur la pile
            push    r13             ; Stock r13 sur la pile
            ...
            pop     r13             ; Restaure r13
            pop     r14             ; Restaure r14
            ret                     ; Retour de sous-programme
```

RCALL – Relative Call to Subroutine – Appel Relatif d'un Sous-programme

Appel d'un sous-programme relatif à l'adresse du **PC** avec une plage de **-2Ko+1** à **+2Ko**. L'adresse de retour (l'instruction après **RCALL**) est stockée sur la Pile. Dans l'assembleur, des étiquettes sont employées au lieu des opérandes relatifs pour simplifier l'écriture. Cette instruction peut adresser la totalité de la mémoire programme. Le pointeur de pile est décrémenté par l'instruction **RCALL**. Cette instruction doit être utilisée pour faire des appels court de sous-programme.

PC = PC + k + 1

Avec : k = -2048 à 2047 (16 bits ou 22 bits).

Instruction :

Code	Opérateurs	Pointeur Pile	Pile	PC	Octet	Cycle
RCALL	k (16 bits) k (22Bits)	PS = PS – 2 PS = PS – 3	PC + 1	PC + k + 1	2	3 4

Code :

Code Instruction	1101	kkkk	kkkk	kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
            rcall   routine         ; Appel de sous-programme relatif
            ...
routine:    push    r14             ; Sauve r14 sur la pile
            ...
            pop     r14             ; Restaure r14
            ret                     ; Retour de sous-programme
```

RET – Return from Subroutine – Retour de Sous-programme

Retour de sous-programme. L'adresse de retour est chargée de la **PILE**. Le pointeur de pile est pré incrément pendant l'exécution de l'instruction **RET**.

PC = PILE (16 bits ou 22 bits)

Instruction :

Code	Opérateurs	Pointeur Pile	PC	Octet	Cycle
RET	(16 bits) (22Bits)	PS = PS + 2 PS = PS + 3	PILE	2	4 5

Code :

Code Instruction	1001	0101	0000	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Appel de sous-programme
                call    routine         ; Appel de sous-programme
                ...
routine:        push    r14             ; Sauve r14 sur la pile
                ...
                pop     r14             ; Restaure r14
                ret                     ; Retour de sous-programme
```

RETI – Return from Interrupt – Retour d'Interruption

Retour d'interruption. L'adresse de retour est chargée de la **PILE**. Le pointeur de pile est pré incrément pendant l'exécution de l'instruction **RET**.

Note : le registre de statut n'est pas automatiquement stocké lors d'une routine d'interruption, il n'est donc pas rétabli au retour d'une interruption, cela doit être traité par votre application.

PC = PILE (16 bits ou 22 bits)

Instruction :

Code	Opérateurs	Pointeur Pile	PC	Octet	Cycle
RETI	(16 bits) (22Bits)	PS = PS + 2 PS = PS + 3	PILE	2	4 5

Code :

Code Instruction	1001	0101	0001	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Retour d'interruption
routine:        push    r0              ; Sauve r0 sur la pile
                ...
                pop     r0              ; Restaure r0
                reti                    ; Retour d'interruption
```

RJMP – Relative Jump – Saut Relatif

Saut relatif à l'adresse du **PC** avec une plage de **-2Ko+1** à **+2Ko**. Dans l'assembleur, des étiquettes sont employées au lieu des opérandes relatifs pour simplifier l'écriture. Cette instruction peut adresser la totalité de la mémoire programme. Cette instruction doit être utilisée pour faire des sauts courts.

PC = PC + k + 1

Avec : k = -2048 à 2047.

Instruction :

Code	Opérateurs	PC	Octet	Cycle
RJMP	k	PC + k + 1	2	2

Code :

Code Instruction	1100	kkkk	kkkk	kkkk

Registre Statut :

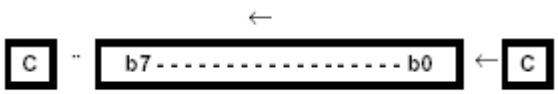
SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Saut relatif
            cpi     r16, $42        ; Compare r16 avec $42
            brne    error           ; Branche si r16 <> $42
            rjmp    ok              ; Branchement inconditionnelle
error:      add     r16, r17        ; Addition r17 à r16
            inc     r16             ; Incrément de r16
ok:         nop                     ; Destination du saut
```

ROL – Rotate Left trough Carry – Rotation à Gauche avec Retenue

Décale tous les bits de **Rd** d'une place à gauche, le drapeau **C** est chargé avec le bit 0 de **Rd** et le bit 7 est chargé avec le drapeau **C**. Cette instruction, combinée avec **LSL**, multiplie efficacement les octets des valeurs signées et non signées par deux.



Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ROL	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	0001	11dd	dddd	dddd

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = Rd3
S = N ⊕ V
V = N ⊕ C
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = Rd7

Exemple :

```
; Rotation à gauche
```

```
                lsl     r18             ; Multiplie r19:r18 par 2
                rol     r19             ; r19:r18 est signé ou non signé sur 2 bits
                brcs    oneenc          ; Branche si C est 1
                ...
oneenc:         nop                     ; Destination du branchement
```

ROR – Rotate Right through Carry – Rotation à Droite avec Retenue

Décale tous les bits de **Rd** d'une place à droite, le drapeau **C** est chargé avec le bit 7 de **Rd** et le bit 0 est chargé avec le drapeau **C**. Cette instruction, combinée avec **ASR**, divise efficacement les octets des valeurs signées par deux et combinée avec **LSR**, divise efficacement les octets des valeurs non signées par deux.

Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ROR	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	1001	010d	dddd	0111

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	X	X	X	X	X

S = N ⊕ V
V = N ⊕ C
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = Rd7

Exemple :

```
; Rotation à Droite
                lsr     r19             ; Divise r19:r18 par 2
                ror     r18             ; r19:r18 est non signé
                brcc    zeroenc1        ; Branche si C est à 0
                asr     r17             ; Divise r17:r16 par 2
                ror     r16             ; r17:r16 est signé
                brcc    zeroenc2        ; Branche si C est à 0
                ...
zeroenc1:       nop                     ; Destination du branchement
                ...
zeroenc2:       nop                     ; Destination du branchement
                ...
```

SBC – Subtract with Carry – Soustraction avec Retenue

Soustrait deux registres **Rd** et **Rr** avec la retenue **C** et place le résultat dans le registre **Rd**.

Rd = Rd - Rr - C

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SBC	Rd, Rr	Rd	PC + 1	2	1

Code :

Code Instruction	0000	10rd	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = Rd3 & Rr3 ! Rr3 & R3 ! R3 & Rd3
S = N ⊕ V
V = Rd7 & Rr7 & R7 ! Rd7 & Rr7 & R7
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0 & Z
C = Rd7 & Rr7 ! Rr7 & R7 ! R7 & Rd7

Exemple :

```
; Soustraction sur 16 bits r1:r0 par r3:r2
        sub     r2, r0          ; Soustraction bas
        sbc     r3, r1          ; Soustraction haut avec retenue
```

SBCI – Subtract Immediate with Carry – Soustraction Immédiate avec Retenue

Soustraction d'une valeur immédiate **K** au registre **Rd** avec la retenue **C** et place le résultat dans le registre **Rd**.

Rd = Rd – K - C

Avec : Rd = 16 à 31 et **K** = 0 à 255.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SBCI	Rd, K	Rd	PC + 1	2	1

Code :

Code Instruction	0100	KKKK	dddd	KKKK

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = Rd3 & Rr3 ! Rr3 & R3 ! R3 & Rd3
S = N ⊕ V
V = Rd7 & Rr7 & R7 ! Rd7 & Rr7 & R7
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0 & Z
C = Rd7 & Rr7 ! Rr7 & R7 ! R7 & Rd7

Exemple :

```
; Soustraction Immédiate $4F23 à r17:r16
        subi    r16, $23        ; Soustraction bas
        sbci    r17, $4F        ; Soustraction haut avec retenue
```

SBI – Set Bit in I/O Register – Active le Bit des Entrée/Sortie

Met le bit '**b**' indiqué dans le registre d'Entrée/Sortie **A**. Cette instruction fonctionne sur la partie inférieure des registres d'Entrée/Sortie - adresse 0 à 31 :

I/O(A, b) = 1

Avec : A = 0 à 31 et **b** = 0 à 7.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SBI	A, b	-	PC + 1	2	1

Code :

Code Instruction	1001	1010	AAAA	Abbb

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Active une Entrée/Sortie
        out     $1E, r0         ; Ecrire l'adresse de l'EEPROM
        sbi     $1C, 0          ; Active le bit d'écriture EECR
        in      r1, $1D         ; Lire la donnée de l'EEPROM
```

SBIC – Skip if Bit in I/O Register is Cleared – Saute si Bit I/O est à 0

Cette instruction évalue un bit '**b**' simple dans une Entrée/Sortie **A** et saute l'instruction suivante si le bit est à 0. Cette instruction fonctionne sur la partie inférieure des Entrée/Sorties - adresse 0 à 31 :

Si I/O(A,b) = 0 alors PC = PC + 2 (ou 3) sinon PC = PC + 1

Avec : A = 0 à 31 et **b** = 0 à 7.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SBIC	A, b	-	PC + 1 PC + 2 PC + 3	2	1 2 3

Code :

Code Instruction	1001	1001	AAAA	Abbb

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Test les bits d'Entrée/Sortie
e2wait:     sbic    $1C,1           ; Saute l'instruction suivante si EEWE = 0
            rjmp    e2wait          ; EEPROM Ecriture non finie
            nop                     ; Continue
```

SBIS – Skip if Bit in I/O Register is Set – Saute si Bit I/O est à 1

Cette instruction évalue un bit '**b**' simple dans une Entrée/Sortie **A** et saute l'instruction suivante si le bit est à 1. Cette instruction fonctionne sur la partie inférieur des Entrée/Sorties - adresse 0 à 31 :

Si I/O(A,b) = 1 alors PC = PC + 2 (ou 3) sinon PC = PC + 1

Avec : A = 0 à 31 et **b** = 0 à 7.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SBIS	A, b	-	PC + 1 PC + 2 PC + 3	2	1 2 3

Code :

Code Instruction	1001	1011	AAAA	Abbb

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Test les bits d'Entrée/Sortie
waitset:    sbis    $10,0       ; Saut l'instruction suivante si bit 0 du Port D = 1
            rjmp    waitset     ; Bit = 0
            nop                 ; Continue
```

SBIW – Subtract Immediate from Word – Soustraction Immédiate 16 bits

Soustraction d'une valeur immédiate **k** (0 - 63) à une paire de registre et place le résultat de la paire de registre dans **Rd1:0**. Cette instruction fonctionne sur 4 paires de registre uniquement et est adapter aux opérations sur les registres d'indicateur. Cette instruction n'est pas disponible dans tous les modèles.

Rd+1:Rd = Rd+1:Rd - k

Avec : Rd = {24, 26, 28 ou 30} et **k** = 0 à 63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SBIW	Rd+1:Rd, k	Rd+1 :Rd	PC + 1	2	2

Code :

Code Instruction	1001	0111	kkdd	kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	**X**	**X**	**X**	**X**	**X**

S = N ⊕ V
V = Rdh7 & R15
N = R15
Z = R15&R14&R13&R12&R11&R10&R9 & R8 & R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = R15 & Rdh7

Exemple :

```
; Soustraction Immédiate
            sbiw    r25:r24, 1      ; Soustraction de 1 à r25:r24
            sbiw    YH:YL, 63       ; Soustraction 63 à Y (pointeur r29:r28)
```

SBR – Set Bits in Register – Active le Bit du Registre

Active les bits dans le registre **Rd** par l'exécution d'un **OU** logique entre le contenu du registre **Rd** et un masque **k** et place le résultat dans **Rd**.

Rd = Rd ! k

Avec : Rd = 16 à 31 et **k** = 0 à 255.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SBR	Rd, k	Rd	PC + 1	2	1

Code :

Code Instruction	0110	kkkk	dddd	kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	X	0	X	X	-

S = N ⊕ V
V = 0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0

Exemple :

```
; Active des bits
        sbr     r16, 3          ; Active le bit 0 et 1 dans r16
        sbr     r17, $F0        ; Active les 4 MSB dans r17
```

SBRC – Skip if Bit in Register is Cleared – Saute si le Bit du Registre est à 0

Cette instruction évalue un bit simple '**b**' dans un registre et saute l'instruction suivante si le bit est à 0.

Si Rr(b) = 0 alors PC = PC + 2 (or 3) sinon PC = PC + 1

Avec : Rr = 0 à 31 et **b** = 0 à 7.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SBRC	Rr, b	-	PC + 1 PC + 2 PC + 3	2	1 2 3

Code :

Code Instruction	1111	110r	rrrr	0bbb

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Test les bits du Registre
        sub     r0, r1          ; Soustraction r1 à  r0
        sbrc    r0, 7           ; Saute si le bit 7 dans r0 est à 0
        sub     r0, r1          ; Exécuté seulement si le bit 7 dans r0 <> 0
        nop                     ; Continue
```

SBRS – Skip if Bit in Register is Set – Saute si le Bit du Registre est à 1

Cette instruction évalue un bit simple '**b**' dans un registre et saute l'instruction suivante si le bit est à 1.

Si Rr(b) = 1 alors PC = PC + 2 (or 3) sinon PC = PC + 1

Avec : Rr = 0 à 31 et **b** = 0 à 7.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SBRS	Rr, b	-	PC + 1 PC + 2 PC + 3	2	1 2 3

Code :

Code Instruction	1111	111r	rrrr	0bbb

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Test les bits du Registre
        sub     r0, r1          ; Soustraction r1 à  r0
        sbrs    r0, 7           ; Saute si le bit 7 dans r0 est à 1
        sub     r0, r1          ; Exécuté seulement si le bit 7 dans r0 <> 1
        nop                     ; Continue
```

SEC – Set Carry Flag – Active la Retenue C = 1

Active la retenue dans le registre **SREG** :

C = 1

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SEC	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0100	0000	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	**1**

Exemple :

```
; Active la Retenue
        sec                     ; Active la retenue
        adc     r0, r1          ; r0 = r0 + r1 + 1
```

SEH – Set Half Carry Flag – Active la Demi Retenue H = 1

Active la demi-retenue dans le registre **SREG** :

H = 1

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SEH	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0100	0101	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	**1**	-	-	-	-	-

Exemple :

```
; Active la demi-retenue
        sec                     ; Active la demi-retenue
```

SEI – Set Global Interrupt Flag – Active les Interruptions I = 1

Active les interruptions dans le registre **SREG** :

I = 1

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SEI	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0111	0000	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	**1**	-	-	-	-	-	-	-

Exemple :

```
; Active les interruptions
        sei                     ; Active les interruptions
        sleep                   ; Entrer en mode sleep, attente une interruption
```

SEN – Set Negative Flag – Active le Signe S = 1

Active le signe dans le registre **SREG** :

S = 1

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SEN	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0100	0010	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	**1**	-	-

Exemple :

```
        add     r2, r19         ; Addition de r19 à r2
        sen                     ; Active le signe Négatif
```

SER – Set all Bits in Register – Active Tous les Bits du Registre

Active tout les bits du registre **Rd** :

Rd = $FF

Avec : Rd = 16 à 31

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SER	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	1110	1111	dddd	1111

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Active tout les bit
        clr     r16             ; Efface r16
        ser     r17             ; Active r17
        out     $18, r16        ; Ecrit 0 dans le Port B
        nop                     ; Attente synchronisation
        out     $18, r17        ; Ecrit 1 dans le Port B
```

SES – Set Signed Flag – Active les Nombres Signés S = 1

Active les nombres signés dans le registre **SREG** :

S = 1

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SES	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0100	0100	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	**1**	-	-	-	-

Exemple :

```
; Active les nombres signés
        add     r2, r19         ; Addition de r19 à r2
        ses                     ; Active les nombres signés
```

SET – Set T Flag – Active le Test T = 1

Active le test dans le registre **SREG** :

T = 1

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SET	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0100	0110	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	**1**	-	-	-	-	-	-

Exemple :

```
; Active le Test
        set                     ; Active le bit de Test
```

SEV – Set Overflow Flag – Active le débordement V = 1

Active le débordement dans le registre **SREG** :

V = 1

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SEV	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0100	0011	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	**1**	-	-	-

Exemple :

```
; Active le débordement
        add     r2, r19         ; Addition de r19 à r2
        sev                     ; Active le débordement
```

SEZ – Set Zero Flag – Active le Zéro Z = 1

Active le zéro dans le registre **SREG** :

Z = 1

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SEZ	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0100	0001	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	**1**	-

Exemple :

```
        add     r2, r19         ; Addition de r19 à r2
        sez                     ; Active le Zéro
```

SLEEP – Mode Sommeil

Cette instruction met le circuit dans le mode de sommeil défini par le registre de contrôle du MCU.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SLEEP	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0101	1000	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Active le mode sommeil
        mov     r0,r11          ; Copie r11 dans r0
        ldi     r16,(1<<SE)     ; Active le mode sommeil
        out     MCUCR, r16
        sleep                   ; Met le MCU en mode sommeil
```

SPM – Store Program Memory – Stock le Programme en Mémoire

SPM peut être employé pour effacer une page dans la mémoire programme, écrire une page dans la mémoire programme (qui est déjà effacé) et mettre des bits de blocage en mode boot. Dans certains modèles, la mémoire programme peut être écrite un mot à la fois, dans d'autres une page entière peut être programmée simultanément après le remplissage d'une zone de page provisoire. Dans tous les cas, la mémoire programme doit être effacée une page à la fois. En effaçant la mémoire programme, la **RAMPZ** et le registre **Z** est employé comme l'adresse de page et la paire de registre de **R1**:**R0** est employée comme des données. En mettant les bits de blocage (boot), la paire de registre de **R1**:**R0** est aussi employée comme des données. Référez-vous à la documentation du modèle pour la description détaillée d'utilisation de **SPM**.

(RAMPZ:Z) = $FFFF – Efface la page de la mémoire programme

(RAMPZ:Z) = R1:R0 Ecrit un mot dans la page de la mémoire programme

(RAMPZ:Z) = R1:R0 Ecrit dans la page buffer provisoire

(RAMPZ:Z) = TEMP Ecrit la page buffer dans la page de la mémoire programme

BLBITS = R1:R0 Active le 'Boot Loader Lock bits'

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SPM	-	-	PC + 1	2	Variable

Code :

Code Instruction	1001	0101	1110	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Programme la mémoire programme, cet exemple montre que SPM écrit une page de données dans
la RAM, le premier emplacement de donné dans la RAM est indiquée par Y, le premier emplacement
de donné dans la Flash est indiqué par le traitement d'erreur de Z.
; Registres utilisés: r0, r1, temp1, temp2, looplo, loophi, spmcrval, (temp1, temp2, looplo, loophi,
spmcrval)
EcriPage:
```

```
.equ        PAGESIZEB = PAGESIZE*2      ; PAGESIZEB est la taille de la page en octet
.org        SMALLBOOTSTART
; Effacement de la page
            ldi     spmcrval, (1<<PGERS) + (1<<SPMEN)
            call    do_spm
; Transfert données de la RAM vers la Flash page provisoire
            ldi     looplo, low(PAGESIZEB)  ; Init boucle variable
            ldi     loophi, high(PAGESIZEB) ; Non requis pour PAGESIZEB<=256
wrloop:     ldr     0, Y+
            ldr     1, Y+
            ldi     spmcrval, (1<<SPMEN)
            call    do_spm
            adiw    ZH:ZL, 2
            sbiw    loophi:looplo, 2            ; Utilise subi pour PAGESIZEB<=256
            brne    wrloop
; Execute l'écriture de page
            subi    ZL, low(PAGESIZEB)      ; Restaure pointeur
            sbci    ZH, high(PAGESIZEB)     ; Non requis pour PAGESIZEB<=256
            ldi     spmcrval, (1<<PGWRT) + (1<<SPMEN)
            call    do_spm
; Lecture arrière et test, optionel
            ldi     looplo, low(PAGESIZEB)  ; Init boucle variable
            ldi     loophi, high(PAGESIZEB) ; Non requis pour PAGESIZEB<=256
            subi    YL, low(PAGESIZEB)      ; Restaure pointeur
            sbci    YH, high(PAGESIZEB)
rdloop:     lpm     r0, Z+
            ldr     1, Y+
            cpse    r0, r1
            jmp     error
            sbiw    loophi:looplo, 2            ; Utilise subi pour PAGESIZEB<=256
            brne    rdloop
; Retour
            ret
do_spm:
; Entrée: spmcrval détermine SPM action, désactive les interruptions si active, stock stratus intemp2, SREG
            cli
; Test pour première SPM complète
wait:       in      temp1, SPMCR
            sbrc    temp1, SPMEN
            rjmp    wait
; SPM temps séquence
            out     SPMCR, spmcrval
            spm
; Restore SREG (et active les interruptions si originellement activé)
            out     SREG, temp2
            ret
```

ST – Store Indirect From Register to Data Space using Index X – Stock Indirect X

Stock un octet indirect d'un registre dans l'espace de données. L'espace de donnée est constitué des registres d'accès rapide, des registres d'Entrée/Sortie et de la **SRAM**. **L'EEPROM** a un espace d'adresse séparé inaccessible directement. L'emplacement de la donnée est indiqué par le registre **X** (16 bits). L'accès de la mémoire est limité au segment de données actuel de 64Ko. Pour avoir accès à un autre segment de données dans des modèles avec plus d'espace de données, le **RAMPX** dans le registre d'Entrée/Sortie doit être changé. Le Registre **X** peut être laissé inchangé par l'opération, ou il peut être incrémenté en post ou en pré décrémentation. Ces particularités sont intéressantes pour l'accès aux tableaux, aux tables et empilement avec l'utilisation du registre **X** comme index. Seule l'octet bas de **X** est mis à jour avec un maximum de 256 octets. L'octet haut de **X** n'est pas modifié par la décrémentation et peut être employé pour d'autres buts. Le résultat des combinaisons suivantes est non défini : **ST X+, r26 ; ST X+, r27 ; ST -X, r26 ; ST -X, r27**.

(X) = Rr [1] ou (X) = Rr, X = X + 1 [2], ou X = X – 1, (X) = Rr [3]

Avec : Rr = 0 à 31 (sauf 26 et 27).

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ST	(X)	-	PC + 1	2	2

Code :

Code Instruction 1	1001	001r	rrrr	1100
Code Instruction 2	1001	001r	rrrr	1101
Code Instruction 3	1001	001r	rrrr	1110

En fonction de l'opération simple [1], de la post-incrémentation [2] ou de la pré-incrémentation [3], le code instruction change.

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Stock indirect X
        clr     r27             ; Efface X bits haut
        ldi     r26, $60        ; X bas à $60
        st      X+, r0          ; Stock r0 dans l'adresse $60 (X post inc)
        st      X, r1           ; Stock r1 dans l'adresse $61
        ldi     r26, $63        ; X bas à $63
        st      X, r2           ; Stock r2 dans l'adresse $63
        st      -X, r3          ; Stock r3 dans l'adresse $62 (X pre dec)
```

ST (STD) – Store Indirect From Register to Data Space using Index Y – Stock Indirect Y

Stock un octet indirectement via **Y** d'un registre d'accès rapide dans l'espace de données. L'espace de données est soit les registres d'accès rapide, soit la mémoire d'Entrée/Sortie, soit la **SRAM** interne. **L'EEPROM** a un espace d'adresse séparé non accessible. L'emplacement de la donnée est indiqué par le registre **Y** (16 bits). L'accès de la mémoire est limité au segment de données sur 64Ko. Pour avoir accès à un autre segment de données dans les modèles avec plus d'espace, la **RAMPY** dans le secteur d'Entrée/Sortie doit être changé. Le Registre **Y** peut être laissé inchangé par l'opération, ou il peut être post-incrémenté ou pré-décrémenté. Ces particularités sont utiles pour l'accès aux tableaux, tables et empilement, avec une restriction à seulement 256 octets, car seul l'octet bas de **Y** est mis à jour. L'octet haut de **Y** n'est pas employé par cette instruction et peut être employé pour d'autres buts. Attention, le résultat de ces combinaisons utilisant le registre **Y** est non défini : **ST Y+, r28 ; ST Y+, r29 ; ST –Y, r28 ; ST –Y, r29**.

(Y) = Rr [1] ou (Y) = Rr, Y = Y + 1 [2], ou Y = Y – 1, (Y) = Rr [3], (Y + q) = Rr [4]

Avec : Rr = 0 à 31 (sauf 28 et 29), **q** = 0 à 63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ST & STD	(Y), q	-	PC + 1	2	2

Code :

Code Instruction 1	1000	001r	rrrr	1000
Code Instruction 2	1001	001r	rrrr	1001
Code Instruction 3	1001	001r	rrrr	1010
Code Instruction 4	10q0	qq1r	rrrr	1qqq

En fonction de l'opération simple [1], de la post-incrémentation [2] ; de la pré-incrémentation [3] ou du décalage dans l'espace d'Entrée/Sortie [4], le code instruction change.

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Stock indirect Y
        clr     r29             ; Efface Y bit haut
        ldi     r28, $60        ; Y bas à $60
        st      Y+, r0          ; Stock r0 dans l'adresse $60(Y post inc)
        st      Y, r1           ; Stock r1 dans l'adresse $61
        ldi     r28, $63        ; Y bas à $63
        st      Y, r2           ; Stock r2 dans l'adresse $63
        st      -Y, r3          ; Stock r3 dans l'adresse $62(Y pre dec)
        std     Y+2, r4         ; Stock r4 dans l'adresse $64
```

ST (STD) – Store Indirect From Register to Data Space using Index Z – Stock Indirect Z

Stock un octet indirectement via **Z** d'un registre d'accès rapide dans l'espace de données. L'espace de données est soit les registres d'accès rapide, soit la mémoire d'Entrée/Sortie, soit la **SRAM** interne. **L'EEPROM** a un espace d'adresse séparé non accessible. L'emplacement de la donnée est indiqué par le registre **Z** (16 bits). L'accès de la mémoire est limité au segment de données sur 64Ko. Pour avoir accès à un autre segment de données dans les modèles avec plus d'espace, la **RAMPY** dans le secteur d'Entrée/Sortie doit être changé. Le Registre **Z** peut être laissé inchangé par l'opération, ou il peut être post-incrémenté ou pré-décrémenté. Ces particularités sont utiles pour l'accès aux tableaux, tables et empilement, avec une restriction à seulement 256 octets, car seul l'octet bas de **Z** est mis à jour. L'octet haut de **Z** n'est pas employé par cette instruction et peut être employé pour d'autres buts. Attention, le résultat de ces combinaisons utilisant le registre **Z** est non défini : **ST +Z, r30 ; ST Z+, r31 ; ST –Z, r30 ; ST –z, r31**.

(Z) = Rr [1] ou (Z) = Rr, Z = Z + 1 [2], ou Z = Z – 1, (Z) = Rr [3], (Z + q) = Rr [4]

Avec : Rd = 0 à 29 (sauf 30 et 31), **q** = 0 à 63.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
ST & STD	(Z), q	-	PC + 1	2	2

Code :

Code Instruction 1	1000	001r	rrrr	0000
Code Instruction 2	1001	001d	rrrr	0001
Code Instruction 3	1001	001d	rrrr	0010
Code Instruction 4	10q0	qq1d	rrrr	0qqq

En fonction de l'opération simple [1], de la post-incrémentation [2] ; de la pré-incrémentation [3] ou du décalage dans l'espace d'Entrée/Sortie [4], le code instruction change.

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Stock indirect Z
        clr     r29             ; Efface Y bit haut
        ldi     r28, $60        ; Y bas à $60
        st      Z+, r0          ; Stock r0 dans l'adresse $60(Z post inc)
        st      Z, r1           ; Stock r1 dans l'adresse $61
        ldi     r28, $63        ; Y bas à $63
        st      Z, r2           ; Stock r2 dans l'adresse $63
        st      -Z, r3          ; Stock r3 dans l'adresse $62(Z pre dec)
        std     Z+2, r4         ; Stock r4 dans l'adresse $64
```

STS – Store Direct to Data Space – Stock Direct dans l'Espace Donnée

Stock un octet directement d'un registre d'accès rapide dans l'espace de données. L'espace de données est soit les registres d'accès rapide, soit la mémoire d'Entrée/Sortie, soit la **SRAM** interne. **L'EEPROM** a un espace d'adresse séparé non accessible. Une adresse 16 bits doit être fournie. L'accès de la mémoire est limité au segment de données de 64Ko. L'instruction **STS** emploie le registre de **RAMPD** pour avoir accès à la mémoire supérieur à 64Ko.

(k) = Rr

Avec : Rr = 16 à 31, **k** = 0 à 65535.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
STS	Rr, k	-	PC + 1	4	2

Code :

Code Instruction	1001	001r	rrrr	0000
Suite instruction	kkkk	kkkk	kkkk	Kkkk

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Charge direct mémoire
        lds     r2, $FF00       ; Charge r2 avec le contenu de l'adresse $FF00
        add     r2, r1          ; Ajout r1 à r2
        sts     $FF00, r2       ; Ecrire en retour
```

SUB – Subtract without Carry – Soustraction sans Retenue

Soustrait deux registres **Rd** et **Rr** sans la retenue et place le résultat dans le registre **Rd**.

Rd = Rd - Rr

Avec : Rd = 0 à 31 et **Rr** = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SUB	Rd, Rr	Rd	PC + 1	2	1

Code :

Code Instruction	0001	10rd	dddd	rrrr

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = Rd3 & Rr3 ! Rr3 & R3 ! R3 & Rd3
S = N ⊕ V
V = Rd7 & Rr7 & R7 ! Rd7 & Rr7 & R7
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0
C = Rd7 & Rr7 ! Rr7 & R7 ! R7 & Rd7

Exemple :

```
; Soustraction sur 16 bits r1:r0 par r3:r2
        sub     r2, r0          ; Soustraction bas
        sbc     r3, r1          ; Soustraction haut avec retenue
```

SUBI – Subtract Immediate – Soustraction Immediate

Soustraction d'une valeur immédiate **K** au registre **Rd** sans la retenue et place le résultat dans le registre **Rd**.

Rd = Rd – K

Avec : Rd = 16 à 31 et **K** = 0 à 255.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SUBI	Rd, K	Rd	PC + 1	2	1

Code :

Code Instruction	0101	KKKK	dddd	KKKK

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	X	X	X	X	X	X

H = Rd3 & Rr3 ! Rr3 & R3 ! R3 & Rd3
S = N ⊕ V
V = Rd7 & Rr7 & R7 ! Rd7 & Rr7 & R7
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0 & Z
C = Rd7 & Rr7 ! Rr7 & R7 ! R7 & Rd7

Exemple :

```
; Soustraction Immédiate $4F23 à r17:r16
        subi    r16, $23        ; Soustraction bas
        sbci    r17, $4F        ; Soustraction haut avec retenue
```

SWAP – Swap Nibbles – Echange Réciproque de Demi Octet

Échange les 4 bits haut avec les 4 bits bas et vice versa d'un registre, instruction utile pour le codage BCD :

R(7:4) = Rd(3:0) et R(3:0) = Rd(7:4)

Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
SWAP	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	1001	010d	Dddd	0010

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Echange haut et bas de Rd
        inc     r1              ; Incrément de r1
        swap    r1              ; Echange haut et bas de r1
        inc     r1              ; Incrément haut de r1
        swap    r1              ; Echange arrière
```

TST – Test for Zero or Minus – Test Zéro ou Négatif

Test si un registre est à zéro ou est négatif. Exécute un **ET** logique entre le registre **Rd** et lui-même. Le registre **Rd** restera inchangé.

Rd = Rd & Rd

Avec : Rd = 0 à 31.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
TEST	Rd	Rd	PC + 1	2	1

Code :

Code Instruction	0010	00dd	dddd	dddd

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	**X**	**0**	**X**	**X**	-

S = N ⊕ V
V = 0
N = R7
Z = R7 & R6 & R5 & R4 & R3 & R2 & R1 & R0

Exemple :

```
; Test si 0 ou négatif
        tst     r0              ; Test r0
        breq    zero            ; Branche si r0 = 0
        ...
zero:   nop                     ; Destination branchement
```

WDR – Watchdog Reset – Remise à Zéro du Chien de Garde

Cette instruction remet à 0 le Timer de garde. Cette instruction doit être exécutée dans un temps limité donné par le pré-diviseur 'Watchdog'. Voir la spécification de matériel de Timer du chien de garde.

Instruction :

Code	Opérateurs	Résultat	PC	Octet	Cycle
WDR	-	-	PC + 1	2	1

Code :

Code Instruction	1001	0101	1010	1000

Registre Statut :

SREG	I	T	H	S	V	N	Z	C
Action	-	-	-	-	-	-	-	-

Exemple :

```
; Remise à 0 du Watchdog
        Wdt                     ; Remise à 0 du Chien de Garde
```

L’assembleur

Ce chapitre décrit l'utilisation de l'assembleur **ATMEL** qui couvre la gamme entière de microcontrôleurs dans la famille **AT90S** et **ATMEGA**. L'assembleur traduit le code source assemblé dans le code d'objet pour être employé avec un simulateur. L'assembleur produit aussi un code **PROMable** et un fichier facultatif **EEPROM** qui peut être programmé directement dans la mémoire programme et la mémoire **EEPROM** d'un microcontrôleur. L'assembleur produit des assignations de code fixées, par conséquent aucune jonction n'est nécessaire. L'assembleur fonctionne sous Windows95 et successeur. De plus, il y a une version de ligne de commande de **MS-DOS**. La version en fenêtres contient une fonction d'aide en ligne couvrant la plupart des commandes ce document.

Début Rapide

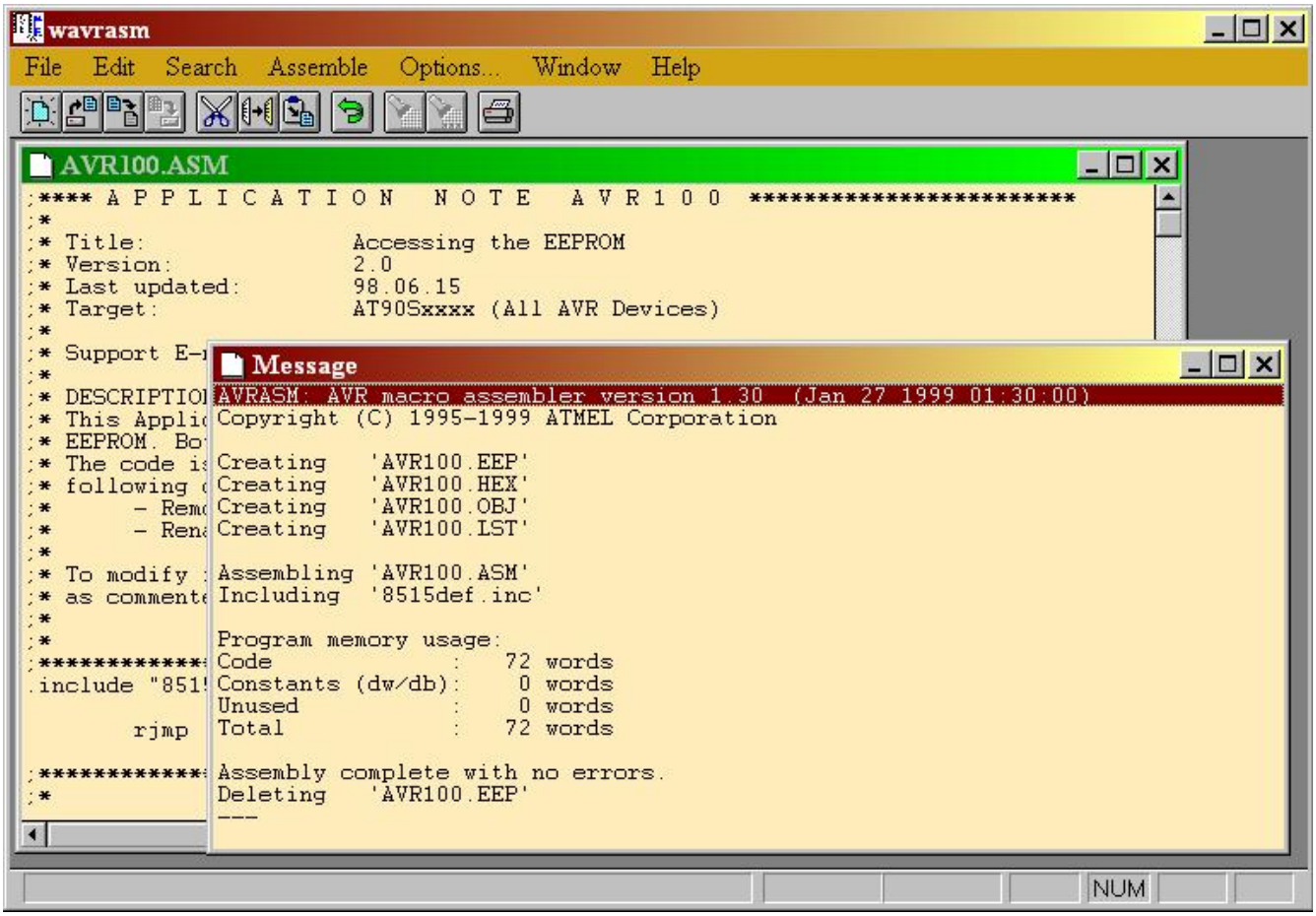
Nous supposons que l'assembleur AVR et tous les fichiers de programme qui viennent avec lui sont correctement installés sur votre ordinateur. Référez-vous s'il vous plaît aux instructions d'installation.

Démarrage

Ouvrir le logiciel **WavrAsm.exe** et commencez l'assemblage en choisissant « File » puis « Open » du menu ou en cliquetant sur la barre d'outils, ouvrez un fichier "avr100.asm". Cela charge le fichier en assembleur dans une fenêtre dite fenêtre rédacteur. Lisez l’entête du programme et jetez un coup d’œil dans le programme mais ne faites pas de changements encore.

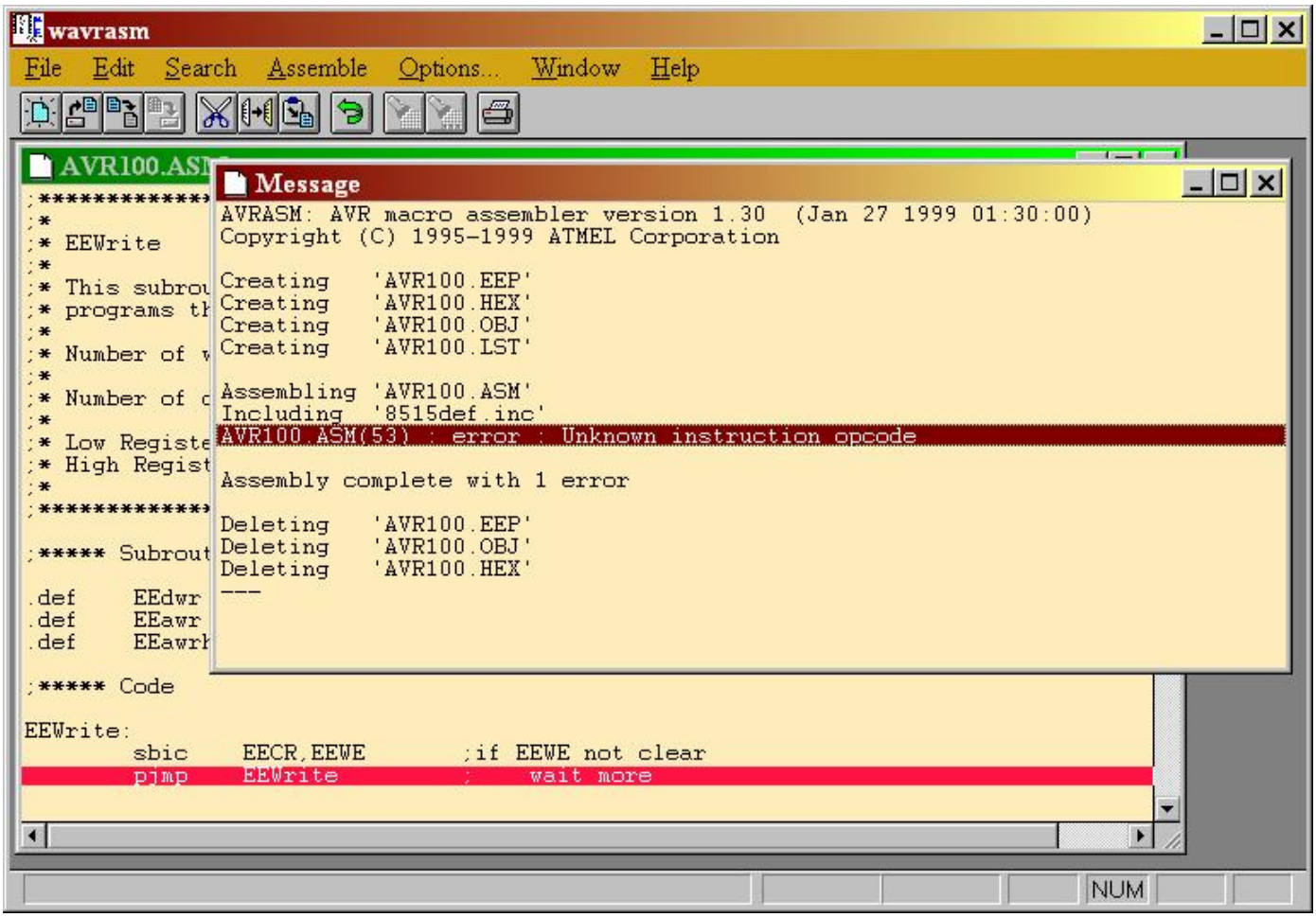
Assemblage de Votre Premier Fichier

Une fois que vous avez regardé le programme, lancé la compilation avec le menu « Assemble ». Une deuxième fenêtre (la fenêtre de Message) apparaît, contenant les messages de compilation et d'erreur. Cette fenêtre chevauchera la fenêtre principale, donc c'est une bonne idée de nettoyer votre espace de travail sur l'écran. Votre écran doit ressembler à cela :



Découverte et Correction d'Erreurs

Dans la fenêtre de Message, il semble que l'assemblage du programme sans bogues (défauts). Mais si on modifie la ligne 53 avec « pjmp », et que l'on assemble de nouveau, il y aura une erreur, elle doit être trouvée et corrigée. Par exemple :



Le premier message d'erreur dans la fenêtre de Message (celui annoncé pour être sur la ligne 53) et pressé sur le bouton gauche de la souris. Remarquez dans le rédacteur la fenêtre, une barre rouge horizontale est montrée sur la ligne 53. Le message d'erreur dit « Unknown instruction opcode ». Voir la figure ci-dessous.

Réassemblage

Une fois toutes les erreurs corrigées, un double-clic sur une erreur (pour activer la fenêtre Rédacteur) ou un clic à l'intérieur de la fenêtre Rédacteur avant de relancer l'assemblage « Assemble » une nouvelle fois. Si vous l'avez fait cela va jusqu'à maintenant, la fenêtre de Message dira que vous êtes couronnés de succès.

Assembler un programme source

L'assembleur travaille sur des fichiers source contenant les mnémoniques d'instruction, les étiquettes et les directives.

Les mnémoniques d'instruction et les directives prennent souvent des opérandes.

Les lignes de code doivent être limitées à 120 caractères.

Chaque ligne d'entrée peut être précédée par une étiquette, qui est une série alphanumérique terminée par un deux-points.

Les étiquettes sont employées comme des cibles pour les sauts et les instructions de branchement et comme des noms de variables dans la mémoire programme et la **SRAM**.

Une ligne d'entrée peut prendre une des quatre des formes suivantes :

1. [étiquette:] directive [opérandes] [; Commentaire]
2. [étiquette:] instruction [opérandes] [; Commentaire]
3. ; Commentaire
4. Ligne vide

Un commentaire à la forme suivante :

; [Texte]

Les articles placés dans des accolades sont facultatifs. Le texte entre le délimiteur de commentaire « **;** » et la fin de ligne **EOL** est ignorée par l'Assembleur.

Les étiquettes, les instructions et les directives sont décrites plus en détail dans les paragraphes suivants.

Example

```
label:      .EQU   var1 = 100        ; Met var1 à 100 (Directive)
            .EQU   var2 = 200        ; Met var2 à 200
test:       rjmp   test              ; Boucle infini (Instruction)
; Ligne de commentaire pur
; Autres commentaires ...
```

Note : Il n'y a aucune restriction en ce qui concerne le placement de colonne d'étiquettes, des directives, des commentaires ou des instructions, mais pour facilité la lecture du programme source, il est préférable de placer les éléments similaires sur les mêmes « colonnes », comme le montre l'exemple.

Directives d'Assemblage

L'Assembleur soutient quelques directives qui ne sont pas traduites directement en « opcodes ». Au lieu de cela, elles sont employées pour ajuster l'emplacement du programme dans la mémoire, définir des macros, initialiser la mémoire etc. On donne une vue d'ensemble des directives dans la table suivante :

Directive	Description
BYTE	Réserve d'octet pour une variable
CSEG	Segment de Code
DB	Définit le(s) octet(s) constant
DEF	Définit un nom symbolique pour un registre
DEVICE	Définit le modèle pour l'assemblage
DSEG	Segment de Données
DW	Définit le(s) mot(s) constant
ENDMACRO	Fin macro
EQU	Attribut un symbole à une expression
ESEG	Segment EEPROM
EXIT	Sortie du fichier
INCLUDE	Importe la source d'un autre fichier
LIST	Met en route la génération de listfile
LISTMAC	Met en route l'extension macro
MACRO	Commence une macro
NOLIST	Arrêt de la génération de listfile
ORG	Origine du programme en mémoire
SET	Déclare un symbole pour une expression

Nous allons maintenant présenter chaque directive d'assemblage avec des exemples.

BYTE – Reserve bytes to a variable - Réserve d'octet pour une variable

La directive **BYTE** réserve des ressources de mémoire dans la **SRAM**. Pour être capable de se référer à l'emplacement réservé, la directive **BYTE** doit être précédée d'une étiquette. La directive prend un paramètre, qui est le nombre d'octets à réserver. La directive peut seulement être employée dans un segment de données (voir des directives **CSEG**, **DSEG** et **ESEG**). Le paramètre est obligatoire et les octets alloués ne sont pas initialisés.

Syntaxe

LABEL: .BYTE expression

Exemple

```
        .DSEG
var1:   .BYTE 1                 ; Réserve 1 octet à var1
table:  .BYTE tab_size          ; Réserve tab_size octets
        .CSEG
        ldi     r30, low(var1)  ; Charge registre Z bas
        ldi     r31, high(var1) ; Charge registre Z haut
        ld      r1, Z           ; Charge VAR1 dans r1
```

CSEG – Code Segment - Segment de Code

La directive **CSEG** définit le début d'un segment de code. Un fichier assembleur peut contenir plusieurs segments de code, qui sont enchaînés dans un seul segment de code après l'assemblage. La directive **BYTE** ne peut pas être employée dans un segment de code. Le type de segment par défaut est le segment de code. Les segments de code ont leur propre compteur d'emplacement qui est un compteur de mot. La directive **ORG** (voir la description plus tard dans ce chapitre) peut être employée pour placer le code et les constantes aux emplacements spécifiques dans la mémoire programme. La directive ne prend pas de paramètres.

Syntax

.CSEG

Example

```
            .DSEG               ; Début du segments de donnée
vartab:     .BYTE 4             ; Réserve 4 octets dans la SRAM
            .CSEG               ; Début du segments de code
const:      .DW 2               ; Ecrire $0002 dans la mémoire programme
            mov   r1, r0        ; Début du programme
```

DB-Define constant byte(s) in program memory or EEPROM memory - Définit le(s) octet(s) constant

La directive **DB** réserve des ressources de mémoire dans la mémoire programme ou la mémoire **EEPROM**. Pour être capable de se référer aux emplacements réservés, la directive **DB** doit être précédée par une étiquette. La directive **DB** prend une liste d'expressions et doit contenir au moins une expression. La directive **DB** doit être placée dans un segments de code ou un segments **d'EEPROM**.

La liste d'expression est un ordre d'expressions, délimitées par des virgules. Chaque expression doit évaluer à un nombre compris entre -128 et 255. Si l'expression évalue un nombre négatif, le complément à 2 des 8 bits du nombre sera placé dans la mémoire programme ou l'emplacement de mémoire **EEPROM**.

Si la directive **DB** est employée dans un segment de code et l'expression list contient plus qu'une expression, les expressions sont réunies deux à deux et placées dans les mots mémoire programme de 16 bits. Si l'expression list contient un nombre impair d'expressions, la dernière expression sera placée dans un mot mémoire programme propre 16 bits, même si la ligne suivante dans le code d'assemblage contient une directive **DB**.

Syntaxe

LABEL: .DB expression list

Example

```
            .CSEG
consts:     .DB 0, 255, 0b01010101, -128, $AA
            .ESEG
eeconst:    .DB $FF
```

DEF - Set a symbolic name on a register - Définit un nom symbolique pour un registre

La directive **DEF** permet aux registres d'être mentionnée par des symboles. Un symbole ainsi défini peut être employé dans le reste du programme pour se référer au registre auquel il est assigné. Un registre peut prendre plusieurs noms symboliques. Un symbole peut être redéfini plus tard dans le programme.

Syntaxe

.DEF Symbol = Registre

Exemple

```
            .DEF   temp = R16
            .DEF   ior = R0
            .CSEG
            ldi    temp, $F0        ; Charge $F0 dans temp (r16)
            in     ior, $3F         ; Lire SREG dans ior (r0)
```

```
        eor     temp, ior           ; Ou exclusif entre temp et ior
```

DEVICE – Define which device to assemble for - Définit le modèle pour l'assemblage

La directive **DEVICE** permet à l'utilisateur de signaler à l'assembleur sur lequel modèle le code doit être exécuté. Si cette directive est employée, un avertissement est affiché si une instruction n'est pas soutenue par le modèle indiqué. Si la taille du segments de code ou le segments **d'EEPROM** est plus grande que l'espace du modèle spécifié, un avertissement est affiché. Si la directive **DEVICE** n'est pas employée, aucun message ne sera affiché et toutes les instructions sont valides, il n'y a aucune restriction de tailles mémoires.

Syntaxe

.DEVICE ATMEGA32

Exemple

```
        .DEVICE ATMEGA32    ; Utilise le modèle ATMEGA32
        .CSEG
        push    r30         ; Place le registre r30 sur la pile
```

DSEG – Data Segment - Segment de Données

La directive **DSEG** définit le début d'un segment de données. Un fichier assembleur peut contenir plusieurs segments de données, qui sont enchaînés dans un seul segment de données dans le code assemblé. Un segment de données seulement contiendra normalement la directive **BYTE** (et les étiquettes).

Les segments de données ont leur propre compteur d'emplacement qui est un compteur d'octet. La directive **ORG** (voir la description plus tard dans ce chapitre) peut être employé pour positionner les variables dans la **SRAM**. La directive ne prend pas de paramètres.

Syntax

.DSEG

Example

```
                .DSEG               ; Début du segment de donnée
var1:           .BYTE 1             ; Réserve 1 octet à var1
table:          .BYTE tab_size      ; Réserve tab_size octets
                .CSEG
                ldi     r30, low(var1)      ; Charge registre Z bas
                ldi     r31, high(var1)     ; Charge registre Z haut
                ld      r1, Z               ; Charge var1 avec r1
```

DW - Define constant word(s) in program memory or EEPROM memory - Définit le(s) mot(s) constant

La directive **DW** réserve des ressources de mémoire dans la mémoire programme ou la mémoire **EEPROM**. Pour être capable de se référer aux emplacements réservés, la directive **DW** doit être précédée par une étiquette. La directive **DW** prend une liste d'expressions et doit contenir au moins une expression. La directive **DB** doit être placée dans un segment de code ou un segment **d'EEPROM**.

La liste d'expression est un ordre d'expressions, délimitées par des virgules. Chaque expression doit évaluer un nombre entre -32768 et 65535. Si l'expression évalue un nombre négatif, le complément à 2 des 16 bits du nombre sera placé dans l'emplacement mémoire programme.

Syntaxe

LABEL: .DW expression list

Exemple

```
                .CSEG
varlist:        .DW     0, $FFFF, 0b1001110001010101, -32768, 65535
                .ESEG
eevar:          .DW     $FFFF
```

ENDMACRO – End macro - Fin macro

La directive **ENDMACRO** définit la fin d'une définition de macro. La directive ne prend pas de paramètres. Voir la directive **MACRO** pour plus d'information sur les macros.

Syntax

.ENDMACRO

Exemple

```
.MACRO      SUBI16                  ; Début de définition d'une macro
            subi    r16, low(@0)    ; Soustraction bas
            sbci    r17, high(@0)   ; Soustraction haut
.ENDMACRO                           ; Fin de la définition de la macro
```

EQU - Set a symbol equal to an expression - Attribut un symbole à une expression

La directive **EQU** assigne une valeur à une étiquette. Cette étiquette peut alors être employée dans des expressions postérieures. Une étiquette assignée à une valeur selon la directive **EQU** est une constante et ne peut pas être changée ou redéfinie.

Syntax

.EQU label = expression

Example

```
            .EQU    io_offset = 0x23
            .EQU    porta = io_offset + 2
            .CSEG                   ; Début du segment
            clr     r2              ; Efface r2
            out     porta, r2       ; Ecrir le Port A
```

ESEG – EEPROM Segment - Segment EEPROM

La directive **ESEG** définit le début d'un segment **d'EEPROM**. Un fichier assembleur peut consister en plusieurs segments **EEPROM**, qui sont enchaînés dans un seul segment **d'EEPROM** après l'assemblage. La directive **BYTE** ne peut pas être employée dans un segment **d'EEPROM**. Les segments **d'EEPROM** ont leur propre compteur d'emplacement qui est un compteur d'octet. La directive **ORG** (voir la description plus tard dans ce chapitre) peut être employé pour positionner le segment dans la mémoire **EEPROM**. La directive ne prend pas de paramètres.

Syntax

.ESEG

Example

```
            .DSEG                   ; Début de segment de donnée
vartab:     .BYTE 4                 ; Réserve 4 octet dans la SRAM
            .ESEG
eevar:      .DW $FF0F               ; Initialise un mot dans l'EEPROM
            .CSEG                   ; Début du segment de code
const:      .DW   2                 ; Ecrit $0002 dans la mémoire programme
            mov   r1, r0            ; Début du programme
```

EXIT - Exit this file - Sortie du fichier

La directive **EXIT** dit à l'assembleur d'arrêter d'assembler le fichier. Normalement, l'assembleur court jusqu'à la fin du fichier **EOF**. Si une directive **EXIT** apparaît dans un fichier inclus, l'assembleur continu à la ligne après la directive **INCLUDE** dans le fichier contenant la directive **INCLUDE**.

Syntaxe

.EXIT

Exemple

```
                .EXIT                   ; Sortie du fichier
```

INCLUDE – Include another file - Importe la source d'un autre fichier

La directive **INCLUDE** dit à l'assembleur de commencer à lire le fichier indiqué. L'assembleur assemble alors le fichier indiqué avant que l'on ne rencontre la fin du fichier **EOF** ou une directive **EXIT**. Un fichier inclus peut contenir aussi des directives **INCLUDE**.

Syntax

.INCLUDE "filename"

Example

```
; iodefs.asm:
                .EQU  sreg = $3F        ; Registre Status
                .EQU  sphigh = $3E      ; Pointeur de Pile haut
                .EQU  splow = $3D       ; Pointeur de Pile bas
                …
; incdemo.asm
                .INCLUDE "iodefs.asm"   ; Inclus I/O definitions
                in    r0, sreg          ; Lire le registre status
```

LIST - Turn the listfile generation on - Met en route la génération de listfile

La directive **LIST** dit à l'assembleur de mettre en route la génération **listfile**. L'assembleur produit un **listfile** qui est une combinaison de code source assemblé, des adresses et des **opcodes**. La génération **listfile** est allumé par défaut. La directive peut aussi être employée avec la directive **NOLIST** pour produire seulement des parties choisies d'un fichier de source d'assemblé.

Syntaxe

.LIST

Exemple

```
                .NOLIST                 ; Désactive la génération listfile
                .INCLUDE "macro.inc"    ; Inclus un fichier macro
                .INCLUDE "const.def"    ; Non visible dans listfile
                .LIST                   ; Active la génération listfile
```

LISTMAC – Turn macro expansion on - Met en route l'extension macro

La directive **LISTMAC** dit à l'assembleur quand une macro est appelée, l'extension d'une macro doit être montrée sur **listfile** produit par l'assembleur. Par défaut, on montre la macro appelée avec les paramètres dans le fichier **listfile**.

Syntax

.LISTMAC

Example

```
.MACRO      MACX                ; Définition d'un exemple de macro
            add   r0, @0
            eor   r1, @1
.ENDMACRO                       ; Fin de la définition de la macro
            .LISTMAC            ; Active l'extension macro
            MACX r2, r1         ; Appel de la macro
```

MACRO – Begin macro - Commence une macro

La directive **MACRO** dit à l'assembleur que c'est le début d'une macro. La directive **MACRO** prend le nom « macro » comme paramètre. Quand le nom de la macro est écrit plus tard dans le programme, la définition macro est employée à la place. Une macro peut prendre jusqu'à 10 paramètres. Ces paramètres sont mentionnés comme **@0-9** dans la définition de la macro. En publiant un appel de macro, on donne les paramètres séparés par une virgule dans la liste. La définition de la macro est terminée avec la directive **ENDMACRO**. Par défaut, on montre l'appel des macros sur le fichier **listfile** produit par l'assembleur.

Pour inclure l'extension macro dans le fichier **listfile**, la directive **LISTMAC** doit être employée. Une macro est marquée avec un symbole **+** dans le champ **opcode** du fichier **listfile**.

Syntax

.MACRO macroname

Example

```
.MACRO      SUBI16                  ; Début de la définition de la macro
            subi    @1, low(@0)     ; Soustraction bas
            sbci    @2, high(@0)    ; Soustraction haut
.ENDMACRO                           ; Fin de la macro
            .CSEG                   ; Début du segment de code
            SUBI16 $1234, r16, r17  ; Soustraction $1234 à r17:r16
```

NOLIST - Turn listfile generation off - Arrêt de la génération de listfile

La directive **NOLIST** dit à l'assembleur d'arrêter la génération du fichier **listfile**. L'assembleur produit normalement un fichier **listfile** qui est une combinaison du code source d'assemblé, des adresses et des **opcodes**. La génération du fichier **listfile** est allumé par défaut, mais peut être mis hors de service en employant cette directive. La directive peut aussi être employée avec la directive **LIST** pour produire seulement des parties choisies d'un fichier de source d'assemblé.

Syntax

.NOLIST ; Désactivation de listfile

Example

```
            .NOLIST                 ; Désactivation de listfile
            .INCLUDE "macro.inc"    ; Inclus un fichier macro
            .INCLUDE "const.def"    ; non visible dans listfile
            .LIST                   ; Active la génération listfile
```

ORG - Set program origin - Origine du programme en mémoire

La directive **ORG** définit le compteur d'emplacement à une valeur absolue. On donne la valeur comme un paramètre de la directive. Si une directive **ORG** est placé dans un segment de données, c'est le compteur d'emplacement **SRAM** qui est mis, si la directive est dans un segment de code, c'est le compteur de mémoire programme qui est mis et si la directive est dans un segment **d'EEPROM**, c'est le compteur d'emplacement **EEPROM** qui est mis. Si la directive est précédée par une étiquette (sur la même ligne de code source), on donnera à l'étiquette la valeur du paramètre. Les valeurs par défaut du code et des compteurs d'emplacement **EEPROM** sont à zéro, tandis que la valeur par défaut du compteur d'emplacement **SRAM** est de 32 (due aux registres rapides occupant les adresses 0 à 31). L'**EEPROM** et l'emplacement **SRAM** sont en octet tandis que l'emplacement de mémoire programme est en mots à 16 bits.

Syntax

.ORG expression

Example

```
            .DSEG                   ; Début du segment de donnée
            .ORG   $67              ; SRAM à l'adresse $67
variable:   .BYTE 1                 ; Réserve 1 octet dans la SRAM
```

```
; adr.$67
                .ESEG                   ; Début du segment d'EEPROM
                .ORG  $20               ; Localisation de l'EEPROM
; compteur
eevar:          .DW   FEFF              ; Initialise un mot
                .CSEG
                .ORG  $10               ; Compteur de Programme (PC)
; 10
                mov   r0, r1
```

SET - Set a symbol equal to an expression - Déclare un symbole pour une expression

La directive **SET** assigne une valeur à une étiquette. Cette étiquette peut alors être employée dans des expressions postérieures. Une étiquette assignée à une valeur selon la directive **SET** peut être changée plus tard dans le programme.

Syntax

.SET label = expression

Example

```
                .SET  io_offset = $23
                .SET  porta = io_offset + 2
                .CSEG                   ; Début du segment de code
                clr   r2                ; Efface r2
                out   porta, r2         ; Ecrit dans le Port A
```

Ceci termine le chapitre sur les directives d'assemblage.

Les Expressions de l'Assembleur

L'Assembleur incorpore des expressions. Les expressions peuvent consister en des opérandes, des opérateurs et des fonctions. En voici le desciptif complet.

Operandes

Les opérandes suivants peuvent être employés :

- L'utilisateur a défini les étiquettes que l'on donne à la valeur du compteur d'emplacement à la place où ils apparaissent.
- L'utilisateur a défini des variables selon la directive de **SET**
- L'utilisateur a défini des constantes selon la directive **EQU**
- Entier constants : selon plusieurs formats, incluant :
  a) - (Défaut) décimal : 10, 255
  b) - Hexadécimal (deux notations) : 0x0A, $0A, 0xFF, $FF
  c) - Valeur binaire : 0b00001010, 0b11111111
- PC - la valeur actuelle du compteur programme en mémoire programme

Fonctions

Les fonctions suivantes sont définies :

- LOW (l'expression) rend l'octet bas d'une expression
- HIGH (expression) rend le deuxième octet d'une expression
- BYTE2 (l'expression) est la même fonction que HIGH
- BYTE3 (l'expression) rend le troisième octet d'une expression
- BYTE4 (l'expression) rend le quatrième octet d'une expression
- LWRD (l'expression) rend les particules 0-15 d'une expression
- HWRD (l'expression) rend les particules 16-31 d'une expression
- PAGE (l'expression) rend les particules 16-21 d'une expression
- EXP2 (l'expression) retourne 2 exposant
- LOG2 (l'expression) rend la partie d'entier (d'expression) log2

Opérateurs

L'Assembleur soutient quelques opérateurs qui sont décrits ici. Les expressions peuvent être incluses dans des parenthèses et telles expressions sont toujours évaluées avant d'être combiné avec les valeurs à l'extérieur des parenthèses.

Unaire Symbole : !

L'opérateur Unaire retourne 1 si l'expression était zéro et retourne 0 si l'expression était non zéro :

```
Exemple :   Ldi   r16, !$F0          ; Charge r16 avec $00
```

Négation Symbole : ~

L'opérateur Négation rend l'expression d'entrée avec tous les bits inversés :

```
Exemple :   Ldi   r16, ~$F0          ; Charge r16 avec $0F
```

Moins Symbole : -

L'opérateur Moins rend la négation arithmétique d'une expression :

```
Exemple :   Ldi   r16, -2            ; Charge –2 ($FE) dans r16
```

Multiplication Symbole : *

L'opérateur Multiplication binaire que rend le produit de deux expressions :

```
Exemple :   Ldi   r30, label * 2     ; Charge r30 avec label multiplier par 2
```

Division Symbole : /

L'opérateur Division binaire rend le quotient entier de l'expression gauche divisée par l'expression juste :

```
Exemple:    Ldi    r30, label / 2        ; Charge r30 avec label diviser par 2
```

Addition Symbole : +

L'opérateur Addition binaire rend la somme de deux expressions :

```
Exemple :   Ldi    r30, c1 + c2          ; Charge r30 avec c1 plus c2
```

Soustraction Symbole : -

L'opérateur Soustraction binaire rend l'expression gauche moins l'expression juste

```
Exemple :   Ldi    r17, c1 - c2          ; Charge r17 avec c1 moins c2
```

Décalage Gauche Symbole : <<

L'opérateur Décalage à Gauche binaire rend l'expression gauche décalé à gauche plusieurs fois fonction de l'expression juste :

```
Exemple :   Ldi    r17, 1<<bitmask       ; Charge r17 avec 1 décalé à gauche de bitmask fois
```

Décalage Droit Symbol: >>

L'opérateur Décalage à Droite binaire rend l'expression gauche décalé à droite plusieurs fois en fonction de l'expression juste :

```
Exemple :   Ldi    r17, c1>>c2           ; Charge r17 avec c1 décalé à droite de c2 fois
```

Plus Petit Que Symbole : <

L'opérateur Plus Petit Que binaire retourne 1 si l'expression signée à gauche est moins grande que l'expression signée de droit, 0 autrement :

```
Exemple :   Ori    r18, bitmask * (c1<c2) + 1 ; OU avec r18
```

Plus Petit ou Equale Symbole : <=

L'opérateur Plus Petit ou Egale binaire retourne 1 si l'expression signée à gauche est moins grande ou Égale à l'expression signée de droit, 0 autrement :

```
Exemple :   Ori    r18, bitmask * (c1<=c2) + 1      ; OU avec r18
```

Plus Grand Que Symbole : >

L'opérateur Plus Grand Que binaire retourne 1 si l'expression signée à gauche est plus grande que l'expression signée à droit, 0 autrement :

```
Exemple :   Ori    r18, bitmask * (c1>c2) + 1 ; OU avec r18
```

Plus Grand ou Equale Symbole : >=

L'opérateur Plus Grand ou Egale binaire retourne 1 si l'expression signée à gauche est plus grande ou Égale à l'expression signée à droit, 0 autrement :

```
Exemple :   Ori    r18, bitmask * (c1>=c2) + 1 ; OU sur r18
```

Equale Symbole : ==

L'opérateur Egale binaire retourne 1 si l'expression signée à gauche est égale à l'expression signée à droit, 0 autrement :

```
Exemple :   Andi   r19, bitmask * (c1==c2)+1 ; ET sur r19
```

Non Equale Symbole : !=

L'opérateur Non Egale binaire retourne 1 si l'expression signée à gauche n'est pas égale à l'expression signée à droit, 0 autrement :

Exemple : .SET flag = (c1!=c2) ; Met le flag avec 1 ou 0

ET entre valeur Symbol : &

L'opérateur ET binaire rend le test ET entre deux expressions :

Exemple : Ldi r18, High(c1&c2) ; Charge r18 avec un ET logique

OU Exclusif Logique Symbol : ^

L'opérateur OU Exclusif binaire retourne le test d'un OU Exclusif entre deux expressions :

Exemple : ldi r18,Low(c1^c2) ;Charge r18 avec un ou exclusif

OU entre deux valeur Symbol : |

L'opérateur OU binaire rend le test OU entre deux expressions :

Exemple : ldi r18, Low(c1|c2) ; Charge 18 avec un OU logique

ET Logique Symbol : &&

L'opérateur ET Logique binaire retourne 1 si les expressions sont tout les deux différent de zéro, 0 autrement :

Exemple : ldi r18, Low(c1&&c2) ; Charge r18 avec un ET logique

OU Logique Symbol : ||

L'opérateur OU Logique binaire retourne 1 si un ou les deux expressions sont différent de zéro, 0 autrement

Exemple : ldi r18, Low(c1||c2) ; Charge r18 avec un OU logique

Ceci clot ce document, bonne programmation.

Jean-Noël