II2 : microcontrôleur

Ce polycopié de cours présente un introduction aux microcontrôleurs. Il s'appuie sur l'exemple d'une famille très utilisée de microcontrôleurs 8bits du fabricant Atmel : la famille AVR.

- La documentation complète des microcontrôleurs AVR est disponible sur le site d'Atmel Www.atmel.com
- l'excellent site des utilisateurs d'AVR http://www.avrfreaks.com/, est très riche, nécessite de s'enregistrer pour avoir accès à l'ensemble des documents
- Enfin http://www.avrtv.com, présente quelques films promotionnels ou des exemples d'applications sur les microcontroleurs AVR

L'interface de développement/simulation AVRstudio 4.13b (et versions ultérieures) est disponible gratuitement sur www.atmel.com (inscription personnelle requise pour le téléchargement)

Le compilateur C AVR-GCC est disponible gratuitement sur http://winavr.sourceforge.net/

1) Introduction

Un microcontroleur = « petit » controleur.

C' est une système informatique de type circuit intégré, généralement utilisé pour contrôler un dispositif physique.

Un microcontrôleur (au contraire d'un microprocesseur) rassemble dans un unique circuit intégré, l'ensemble d'un système à microprocesseur : CPU, RAM, ROM, circuits d'interfaces diverses, etc....

Comme un microprocesseur, un microcontrôleur exécute séquentiellement une suite d'instructions (un programme) en langage assembleur.. Ce programme peut être écrit directement dans le langage assembleur du microcontrôleur (de plus en plus rare), mais la plupart du temps, le programme a été développé en langage de haut niveau (Langage C principalement) puis traduit en assembleur par un Compilateur C.

Microcontrôleur	**Microprocesseur**
Peut fonctionner seul	Nécessite mémoires, chipset, disque dur, interfaces,....
De 6 à >100 broches	>500 ou 1000 broches (en 2008)
Plus de 100 fournisseurs/architectures	<10 fournisseurs(Intel, AMD, IBM, SUN .), moins de 5 architectures subsistent (i386, Power, Sparc,
Plus de 100 à 1000 références par fournisseur	Qcq 10aines de références par fournisseur
Fréquence max 20Mhz~200Mhz	Qcq GHz

Faible à moyenne puissance de calcul	Tres grande puissance de calcul
Très faible consommation (<mW ou µW)	Forte consommation >qcq W
Coût très faible 1€ (<0.1€~10€)	>100€
4,**8** , 16, 32 bits	32 ou **64**bits

Le faible coût des microcontrôleurs, l'énorme variété d'applications dédiées, associés à l'intégration de tout ce qui est nécessaire à leur fonctionnement dans un unique boîtier ont généralisé leurs usage :

- En 2008 : plus de 5Milliards de microcontrôleurs 8bits seront vendus (~5Md$).
- On estime à ~300 le nombre de microcontrôleurs par personne dans les pays développés :
  - Voiture (>50 microcontrôleurs/voiture : capteurs pression pneus, Abs/ESP, injection, contrôle pollution, radar recul, lèves vitres, tableau de bord, climatisation, radio, gps, alarme, boite vitesse, bip, clefs, télépéage, bus CAN
  - Ordinateurs (>20 microcontrôleurs/PC : alimentation, gestion alimentation, ventilateur, Usb, disque dur, souris, clavier, écran (plusieurs),
  - téléphone cellulaire (>5),
  - Jouets pour enfants, brosse à dents, gps, lecteur mp3, ampoules basse consommation, cafetière, fours, lave vaisselle, box adsl, routeurs, scanner (>5), imprimantes(>5), Appareil photos (>5), camera, cartes de paiement, RFID, chargeur de téléphone/batterie.... etc.....

Voir le film "Special : the story of AVR" sur avrtv.com

En conclusion les microcontrôleurs sont partout mais invisibles : ils équipent les systèmes embarqués :
- contraintes de coût et de consommation (alimentation autonome par pile/micropile/ induction)
- application dédiée => la puissance de calcul, taille de la mémoire, nombre d'E/S sont ajustés à l'application cible en vue de réduire les coûts.
- E/S variées : logiques, analogiques,
- il n'y a pas forcement d'interface homme machine ou celle ci peut être particulièrement réduite : *sur un ABS de voiture le « clavier » n'a qu'une touche actionnable avec le pied :-)*

1.1 Elements d'architecture interne

La CPU « Central Processing Unit » ou « unité centrale de traitement » d'un microcontrôleur/microprocesseur exécute séquentiellement des instructions, ces instructions manipulent des données (au sens général). Il est donc nécessaire d' « alimenter » cette CPU avec des Instructions à exécuter et des données sur lesquelles exécuter ces instructions. L'exécution des instructions est cadencé par une horloge (CPU clock), souvent fournie par un oscillateur à quartz.

Dans les 3 schémas suivants c'est la « Central Processing Unit » (CPU) qui exécute ces instructions. Dans le cas d'un microprocesseur, il n'y a que la CPU dans le microprocesseur (par abus de langage on assimile souvent CPU = Microprocesseur). Au contraire un microcontrôleur contient la CPU, de la mémoire (Programmes et

Données) et des interfaces/ports d'entrées/sorties (I/O ports) (

Le CPU et la/les zones mémoires sont reliés par un (plusieurs) BUS qui transportent respectivement des instructions ou des données (Architecture Harvard), ou les deux indifféremment (architecture Von Neumann).

Le BUS se compose classiquement de 3 types de signaux :
- Bus d'adresse (unidirectionel) qui transporte l'adresse mémoire dont la CPU lit ou écrit le contenu.
- Bus de données (bidirectionnel) qui transporte la donnée de/vers la memoire
- Bus de contrôle(unidirectionnel) qui rassemble les signaux logiques de contrôle de l'échange.

Dans ces 3 exemples, la CPU doit donc aller en permanence chercher une instruction à exécuter, puis (souvent mais pas toujours) une donnée sur laquelle cette instruction devra « travailler »

2 grande classes d'architectures de processeurs (synonyme d'organisation interne) coéxistent :
- L'architecture Von neumann, les données et les programmes sont stockés dans la même zone mémoire, c'est une architecture plutot ancienne.
- l'architecture Harvard où les données et les programmes sont stockés dans 2 zones mémoires différentes

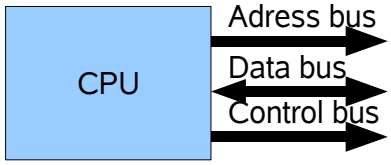


Illustration 1: Microprocesseur

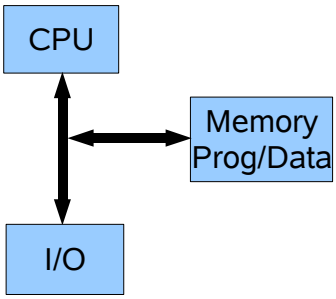


Illustration 2: Microcontroleur architecture Von Neumann

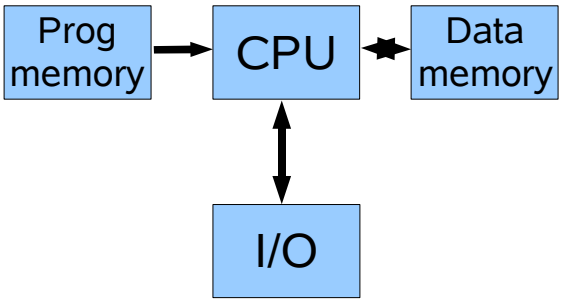


Illustration 1: Microcontrôleur architecture Harvard

Avec l'augmentation des capacités de traitement des CPU, il est devenu possible de traiter 1 instruction par cycle d'horloge. Il devenait alors impossible dans l'architecture Von Neumann d'accéder à la mémoire pour charger cette instruction ET dans le même cycle de charger la donnée à traiter. Les microcontrôleurs/microprocesseurs dont l'architecture est récente utilisent tous une architecture de type Harvard.

1.2 Quelques définitions

On note **MIPS** le nombre de millions d'instructions exécutables par seconde (Million Instruction Per Second).

Par exemple, un microcontrôleur capable d'exécuter 1 instruction par cycle d'horloge CPU (CPU clock) pourra exécuter 20MIPS si il est cadencé par un oscillateur à quartz à 20MHz.

On appelle « microcontroleur 8bits » un microcontrôleur dont la CPU traite des données codées en binaire sur 8 bits.

Attention : les instructions assembleurs sont sur 8 bits mais on peut faire des additions 32 bits en C même sur une microcontroleur 8 Bits : c'est le compilateur qui découpera cette addition C 32 bits en 4 additions assembleur 8 bits avec propagation des retenues intermédiaires.

On appelle **RISC** (Reduced Instruction Set CPU) un CPU qui a un jeu d'instruction réduit (<100 instructions typiquement). Par opposition à **CISC** (Complex Instruction Set CPU) qui peut avoir beaucoup plus de 100 instructions.

L'idée étant qu'avec peu d'instruction, les instructions sont plus simples et s'exécutent plus rapidement. Même si il faut 3 instructions « simples » sur une CPU Risc au lieu de 1 instruction sur une CPU CISC, la rapidité d'exécution d'une instruction RISC fera plus que compenser leur nombre.

Atmel AVR = microcontroleurs 8 bits , RISC à architecture Harvard.

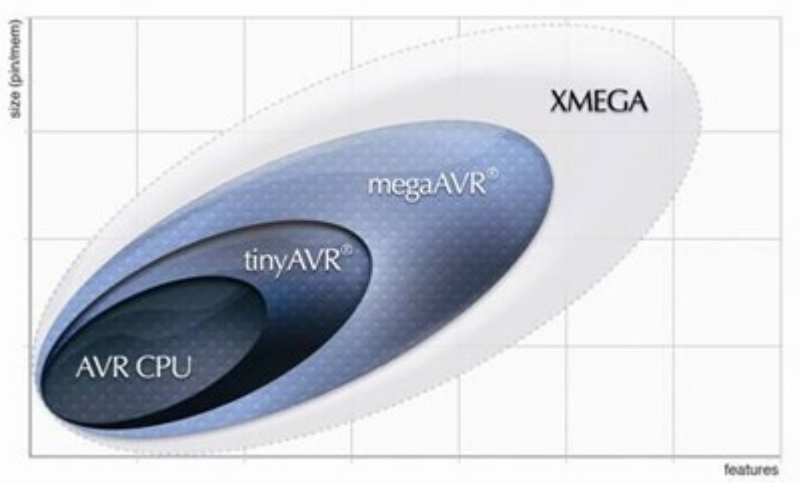
Il exitste bien d'autres architectures microcontroleurs : PIC microchip, 8051 intel, 68HCxx motorola/freescale.... La suite de ce polycopié se consacre à la famille AVR 8.

II2 : microcontrôleur

2) Famille AVR

La famille AVR 8bits (ATMEL) regroupe une centaine de composants, chacun disponible dans plusieurs types de boîtiers physique (package). La famille AVR peut être découpées en 3 sous familles principales :

- TinyAVR (8 à 20 broches)
- Méga AVR (32 à 100 broches)
- xMega AVR (44 à 100 broches) NEW 2008

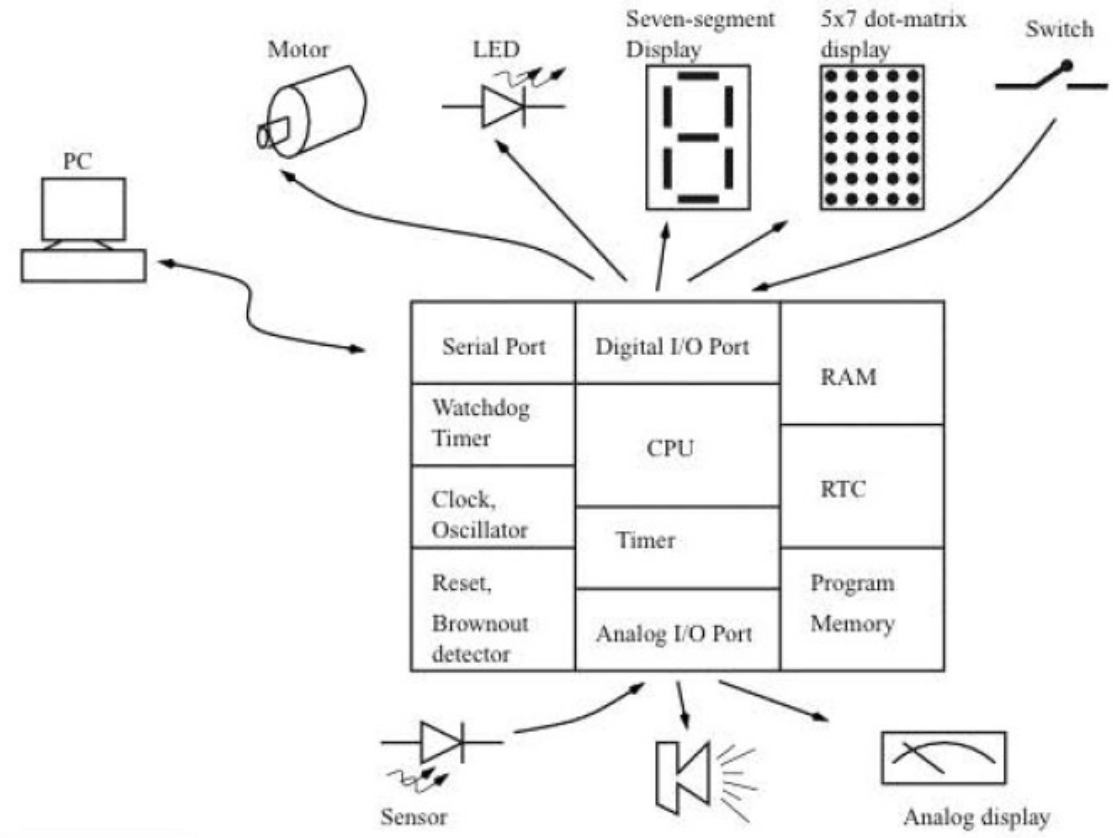


Complétés par des familles développées des marchés spécifiques :

Automotive AVR : microcontroleurs AVR dédiés industrie automobile
Lightning AVR : microcontroleurs AVR dédiés éclairage intelligent
LCD AVR : microcontroleursAVR dédiés contrôle d'affichage LCD
USB AVR : microcontroleurs AVR dédiés bus USB
Zlink AVR : microcontroleurs AVR avec bus ZigBee (Radio 2.4Ghz)

Ces composants nombreux, se différencient par la quantité de mémoires (Données/Programmes) et le nombre et les types de périphériques qu'ils intègrent :

- Mémoire FLASH (programmes) de 64 (Tiny11) a 256koctets (ATMEGA256)
- Mémoire RAM (données) : de 0 (Tiny11) à 16kO (xMEGA)
- jusqu'à 8 ports de 8bits en E/Sortie Génériques
- Mémoire EEPROM de taille variable
- le nombre de PWM permettant de commander des moteurs,
- Le nombre de timers/compteurs
- la variété des liaisons Séries (UART,SPI,I2C, ...)
- la présence de convertisseurs analogique/numérique
- .....



Par contre les microcontroleurs de la famille AVR partagent de très nombreuses caractéristiques communes :

- Les instructions (assembleurs) ont une taille fixe de 16bits (ou 32 bits quelquefois)
- La plupart des instructions (assembleurs) sont exécutées en 1 cycle d'horloge (sauf ADDW et MULS et les instructions de sauts)
- La CPU contient 32 registres 8bits de travail. C'est sur ces registres que peuvent porter les instructions arithmétiques et logiques.
- 64 registres 8bits permettent de contrôler/configurer les périphériques et le coeur du microcontroleur.

Parametric Table AVR

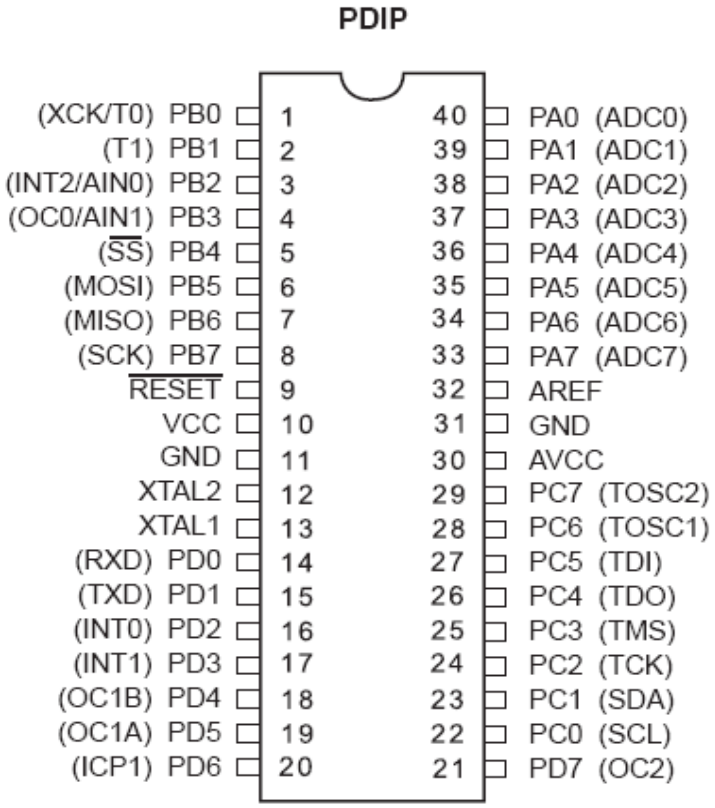
Device	Flash (Kbytes)	EEPROM (Kbytes)	SRAM (Bytes)	Max I/O Pins	F.max (MHz)	Vcc (V)	10-bit A/D Channels	Analog Comparator	16-bit Timers	8-bit Timer	Brown Out Detector	External Interrupts	Hardware Multiplier	Total Interrupts	In Situ Prog.	On Chip Oscillator	PWM Channels	RealTime clock	SPI Interface	2-Wire Interface	UART	Watchdog	Green Packages
ATmega128	128	4	4096	53	16	2.7-5.5	8	Yes	2	2	Yes	8	Yes	34	Yes	Yes	8	Yes	1	Yes	2	Yes	MLF (VQFN) 64 TQFP 64
ATmega1280	128	4	8192	86	16	1.8-5.5	16	Yes	4	2	Yes	32	Yes	57	Yes	Yes	16	Yes	1+USART	Yes	4	Yes	
ATmega1281	128	4	8192	54	16	1.8-5.5	8	Yes	4	2	Yes	17	Yes	48	Yes	Yes	9	Yes	1+USART	Yes	2	Yes	
ATmega16	16	0.5	1024	32	16	2.7-5.5	8	Yes	1	2	Yes	3	Yes	20	Yes	Yes	4	Yes	1	Yes	1	Yes	MLF (VQFN) 44 PDIP 40 TQFP 44
ATmega162	16	0.5	1024	35	16	1.8-5.5	--	Yes	2	2	Yes	3	Yes	28	Yes	Yes	6	Yes	1	--	2	Yes	MLF (VQFN) 44 PDIP 40 TQFP 44
ATmega164P	16	0.5	1024	32	20	1.8-5.5	8	Yes	1	2	Yes	32	Yes	31	Yes	Yes	6	Yes	1+USART	Yes	2	Yes	MLF (VQFN) 44 PDIP 40 TQFP 44
ATmega165	16	0.5	1024	54	16	1.8-5.5	8	Yes	1	2	Yes	17	Yes	23	Yes	Yes	4	Yes	1+USI	USI	1	Yes	MLF (VQFN) 64 TQFP 64
ATmega165P	16	0.5	1024	54	16	1.8-5.5	8	Yes	1	2	Yes	17	Yes	23	Yes	Yes	4	Yes	1+USI	USI	1	Yes	MLF (VQFN) 64 TQFP 64
ATmega168	16	0.5	1024	23	20	1.8-5.5	6/8	Yes	1	2	Yes	26	Yes	26	Yes	Yes	6	Yes	1+USART	Yes	1	Yes	MLF (VQFN) 32 PDIP 28 TQFP 32
ATmega168P	16	0.5	1024	23	20	1.8-5.5	8	Yes	1	2	Yes	26	Yes	26	Yes	Yes	6	Yes	1+USART	Yes	1	Yes	MLF (VQFN) 32 PDIP 28 TQFP 32
ATmega2560	256	4	8192	86	16	1.8-5.5	16	Yes	4	2	Yes	32	Yes	57	Yes	Yes	16	Yes	1+USART	Yes	4	Yes	TQFP 100
ATmega2561	256	4	8192	54	16	1.8-5.5	8	Yes	4	2	Yes	17	Yes	48	Yes	Yes	9	Yes	1+USART	Yes	2	Yes	MLF (VQFN) 64 TQFP 64
ATmega32	32	1	2048	32	16	2.7-5.5	8	Yes	1	2	Yes	3	Yes	19	Yes	Yes	4	Yes	1	Yes	1	Yes	MLF (VQFN) 44 PDIP 40 TQFP 44
ATmega324P	32	1	2048	32	20	1.8-5.5	8	Yes	1	2	Yes	32	Yes	31	Yes	Yes	6	Yes	1+USART	Yes	2	Yes	MLF (VQFN) 44 PDIP 40 TQFP 44
ATmega325	32	1	2048	54	16	1.8-5.5	8	Yes	1	2	Yes	17	Yes	23	Yes	Yes	4	Yes	1+USI	USI	1	Yes	MLF (VQFN) 64 TQFP 64
ATmega3250	32	1	2048	69	16	1.8-5.5	8	Yes	1	2	Yes	17	Yes	32	Yes	Yes	4	Yes	1+USI	USI	1	Yes	TQFP 64 TQFP 100
ATmega3250P	32	1	2048	69	20	1.8-5.5	8	Yes	1	2	Yes	17	Yes	32	Yes	Yes	4	Yes	1+USI	USI	1	Yes	TQFP 100
ATmega325P	32	1	2048	54	20	1.8-5.5	8	Yes	1	2	Yes	17	Yes	23	Yes	Yes	4	Yes	1+USI	USI	1	Yes	MLF (VQFN) 64 TQFP 64
ATmega328P	32	1	2048	23	20	1.8-5.5	8	Yes	1	2	Yes	26	Yes	26	Yes	Yes	6	Yes	1+USART	Yes	1	Yes	MLF (VQFN) 32 PDIP 28 TQFP 32
ATmega48	4	0.25	512	23	20	1.8-5.5	6/8	Yes	1	2	Yes	26	Yes	26	Yes	Yes	6	Yes	1+USART	Yes	1	Yes	MLF (VQFN) 32 PDIP 28 TQFP 32 MLF (VQFN) 28
ATmega48P	4	0.25	512	23	20	1.8-5.5	8	Yes	1	2	Yes	26	Yes	26	Yes	Yes	6	Yes	1+USART	Yes	1	Yes	MLF (VQFN) 32 PDIP 28 TQFP 32 MLF (VQFN) 28
ATmega64	64	2	4096	54	16	2.7-5.5	8	Yes	2	2	Yes	8	Yes	34	Yes	Yes	8	Yes	1	Yes	2	Yes	MLF (VQFN) 64 TQFP 64
ATmega640	64	4	8192	86	16	1.8-5.5	16	Yes	4	2	Yes	32	Yes	57	Yes	Yes	16	Yes	1+USART	Yes	4	Yes	
ATmega644	64	2	4096	32	20	1.8-5.5	8	Yes	1	2	Yes	32	Yes	31	Yes	Yes	6	Yes	1+USART	Yes	1	Yes	MLF (VQFN) 44 PDIP 40 TQFP 44
ATmega644P	64	2	4096	32	20	1.8-5.5	8	Yes	1	2	Yes	32	Yes	31	Yes	Yes	6	Yes	1+USART	Yes	2	Yes	MLF (VQFN) 44 PDIP 40 TQFP 44
ATmega645	64	2	4096	54	16	1.8-5.5	8	Yes	1	2	Yes	17	Yes	23	Yes	Yes	4	Yes	1+USI	USI	1	Yes	MLF (VQFN) 64 TQFP 64
ATmega6450	64	2	4096	69	16	1.8-5.5	8	Yes	1	2	Yes	17	Yes	32	Yes	Yes	4	Yes	1+USI	USI	1	Yes	TQFP 64 TQFP 100
ATmega8	8	0.5	1024	23	16	2.7-5.5	6/8	Yes	1	2	Yes	2	Yes	18	Yes	Yes	3	Yes	1	Yes	1	Yes	MLF (VQFN) 32 PDIP 28 TQFP 32
ATmega8515	8	0.5	512	35	16	2.7-5.5	--	--	1	1	Yes	3	Yes	16	Yes	Yes	3	--	1	--	1	Yes	MLF (VQFN) 44 PDIP 40 TQFP 44
ATmega8535	8	0.5	512	32	16	2.7-5.5	8	Yes	1	2	Yes	3	Yes	20	Yes	Yes	4	--	1	Yes	1	Yes	MLF (VQFN) 44 PDIP 40 TQFP 44
ATmega88	8	0.5	1024	23	20	1.8-5.5	6/8	Yes	1	2	Yes	26	Yes	26	Yes	Yes	6	Yes	1+USART	Yes	1	Yes	MLF (VQFN) 32 PDIP 28 TQFP 32
ATmega88P	8	0.5	1024	23	20	1.8-5.5	8	Yes	1	2	Yes	26	Yes	26	Yes	Yes	6	Yes	1+USART	Yes	1	Yes	MLF (VQFN) 32 PDIP 28 TQFP 32
Device	**Flash (Kbytes)**	**EEPROM (Kbytes)**	**SRAM (Bytes)**	**Max I/O Pins**	**F.max (MHz)**	**Vcc (V)**	**10-bit A/D Channels**	**Analog Comparator**	**16-bit Timers**	**8-bit Timer**	**Brown Out Detector**	**External Interrupts**	**Hardware Multiplier**	**Total Interrupts**	**In Situ Prog.**	**On Chip Oscillator**	**PWM Channels**	**RealTime clock**	**SPI Interface**	**2-Wire Interface**	**UART**	**Watchdog**	**Green Packages**
ATtiny12	1	0.0625	--	6	8	1.8-5.5	--	Yes	--	1	Yes	1	--	5	Yes	Yes	--	--	--	--	--	Yes	PDIP 8 SOIC (209mil) 8
ATtiny13	1	0.0625	64B + 32 reg	6	20	1.8-5.5	4	Yes	--	1	Yes	6	No	9	Yes	Yes	2	--	--	--	--	Yes	MLF (WQFN) 20 PDIP 8 SOIC (209mil) 8
ATtiny2313	2	0.125	128	18	20	1.8-5.5	--	Yes	1	1	Yes	2	--	8	Yes	Yes	4	--	USI	USI	1	Yes	MLF (WQFN) 20 PDIP 20 SOIC (300mil) 20
ATtiny24	2	0.125	128	12	20	1.8-5.5	8	Yes	1	1	Yes	12	--	17	Yes	Yes	4	--	USI	USI	--	Yes	MLF (WQFN) 20 PDIP 14 SOIC 14
ATtiny25	2	0.125	128	6	20	1.8-5.5	4	Yes	--	2	Yes	7	--	15	Yes	Yes	4	--	USI	USI	--	Yes	MLF (WQFN) 20 PDIP 8 SOIC (209mil) 8
ATtiny26	2	0.125	128	16	16	2.7-5.5	11	Yes	--	2	Yes	1	--	11	Yes	Yes	2	--	USI	USI	--	Yes	MLF (VQFN) 32 PDIP 20 SOIC (300mil) 20
ATtiny261	2	0.125	128	16	20	1.8 - 5.5	11	Yes	1	2	Yes	2	--	19	Yes	Yes	2	--	Yes	USI	--	Yes	PDIP 20 SOIC (300mil) 20 SOIC 20
ATtiny28L	2	--	32	11	4	1.8-5.5	--	Yes	--	1	--	2(+8)	--	5	--	Yes	--	--	--	--	--	Yes	
ATtiny44	4	0.25	256	12	20	1.8-5.5	8	Yes	1	1	Yes	12	--	17	Yes	Yes	4	--	USI	USI	--	Yes	MLF (WQFN) 20 PDIP 14 SOIC 14
ATtiny45	4	0.25	256	6	20	1.8-5.5	4	Yes	--	2	Yes	7	--	15	Yes	Yes	4	--	USI	USI	--	Yes	MLF (WQFN) 20 PDIP 8 SOIC (209mil) 8
ATtiny461	4	0.25	256	16	20	1.8 - 5.5	11	Yes	1	2	Yes	2	--	19	Yes	Yes	2	--	Yes	USI	--	Yes	MLF (VQFN) 32 PDIP 20 SOIC 20
ATtiny84	8	0.5	512	12	20	1.8-5.5	8	Yes	1	1	Yes	12	--	17	Yes	Yes	4	--	USI	USI	--	Yes	MLF (WQFN) 20 PDIP 14
ATtiny85	8	0.5	512	6	20	1.8-5.5	4	Yes	--	2	Yes	7	--	15	Yes	Yes	4	--	USI	USI	--	Yes	MLF (WQFN) 20 PDIP 8 SOIC (209mil) 8
ATtiny861	8	0.5	512	16	20	1.8 - 5.5	11	Yes	1	2	Yes	2	--	19	Yes	Yes	2	--	Yes	USI	--	Yes	MLF (VQFN) 32 PDIP 20 SOIC 20

3) Présentation ATMega 16

L'ATméga16 est un composant intermédiaire de la famille AVR 8 de taille raisonnable (40 broches), qui existe en boîtier DIP et qui est très bien fourni en périphériques.

- Disponible en boîtier DIP40
- 32 lignes d'entrée/sorties
- 512 octets de SRAM
- 16Ko de mémoire Flash
- 512 octets EEPROM

Pinout ATmega16



Le brochage (DIP) est donnée sur la figure ci dessus (vue du dessus, noter l'encoche supérieure permettant de repérer l'orientation) . Les noms entre parenthèses correspondent à une $2^{ème}/3^{ème}$ fonctionnalité de la broche. Ces fonctionnalités sont configurables (activables) de manière logicielle. Les broches indispensables pour cabler un ATMEGA

- VCC +5V (fonctionne jusqu'à 2.7V)
- GND masse
- RESET Réinitialisation du microcontroleur (actif a l'état BAS, A CONNECTER A +5V pour fonctionnement)
- XTAL1/XTAL2 Quartz ou entrée d'horloge (*freq*<16MHz sous 5V)

Pour citer quelques unes des fonctionnalités

- PA0,PA1...PA7 Port A : Entrées Sorties
- PBx,PCx,PDx Port B,C,D : 8 Entrées Sorties pour chaque port

- ADC0, ADC7 Seconde fonctionnalité du port A = Entrée analogique
- /SS,MOSI,MISO,SCLK Entrée/sortie série synchrone SPI (PB4/PB7)
- INT0/INT1/INT2 Entrée d'interruption du processeur par un événement externe
- RXD/TXD Liaison série Asynchrone RS232 (communication PC)
- ...

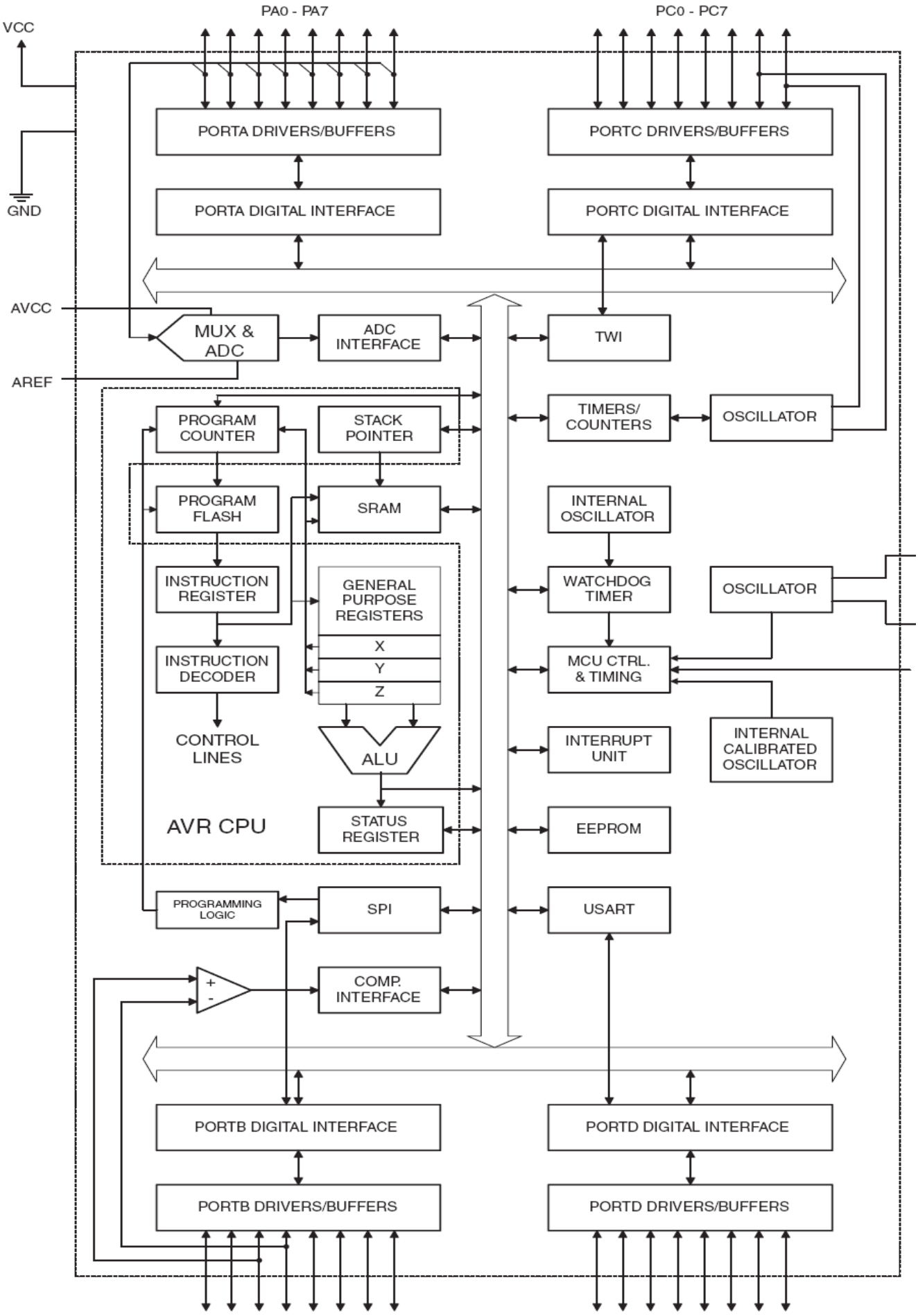
L'architecture interne du microcontroleur est donnée figure suivante. Elle est représentative des microcontroleurs AVR. Suivant le microcontroleur (ATMEGA, tiny) certains périphériques disparaîtront ou apparaîtront, mais sans changer la structure interne.

Au centre à gauche : la CPU, c'est le coeur du microcontroleur, c'est la CPU qui exécute les instructions. On y reconnaît l'architecture Harvard avec ses mémoires données (SRAM) et programme (PROGRAM FLASH) séparées.

En haut et en bas : les 4 ports d'entrées sorties A B C D : On peut y remarquer les doubles fonctionnalités des broches. Par exemple les 8 broches du port A sont également connectées au bloc de conversion analogique numérique (MUX&ADC).

A droite les blocs fonctionnels de contrôles du microcontrôleur et de certains périphériques :
- TIMER,
- clock source,
- contrôleur d'interruption (INTERRUPT)
- liaisons séries TWI, USART, SPI
- ....

Enfin au milieu : un bus permettant le transfert de données entre les périphériques/mémoires et la CPU


PA0 - PA7
PC0 - PC7
VCC
GND
PORTA DRIVERS/BUFFERS
PORTC DRIVERS/BUFFERS
PORTA DIGITAL INTERFACE
PORTC DIGITAL INTERFACE
AVCC
AREF
MUX & ADC
ADC INTERFACE
TWI
PROGRAM COUNTER
STACK POINTER
TIMERS/ COUNTERS
OSCILLATOR
PROGRAM FLASH
SRAM
INTERNAL OSCILLATOR
INSTRUCTION REGISTER
GENERAL PURPOSE REGISTERS
X
Y
Z
WATCHDOG TIMER
OSCILLATOR
INSTRUCTION DECODER
MCU CTRL. & TIMING
CONTROL LINES
ALU
INTERRUPT UNIT
INTERNAL CALIBRATED OSCILLATOR
AVR CPU
STATUS REGISTER
EEPROM
PROGRAMMING LOGIC
SPI
USART
COMP. INTERFACE
PORTB DIGITAL INTERFACE
PORTD DIGITAL INTERFACE
PORTB DRIVERS/BUFFERS
PORTD DRIVERS/BUFFERS

4) ORGANISATION MEMOIRE

Les CPU AVR contiennent plusieurs types de mémoires :

- la mémoire dédiée aux programmes, cette mémoire est de type FLASH, elle conserve les programmes (ici le programme qu'exécutera le microcontrôleur) même en absence d'alimentation.
- la mémoire RAM qui contient les données, cette mémoire est de type SRAM. La SRAM perd les données en absence d'alimentation.
- La mémoire EEPROM, qui contient des données mais conserve ces données si l'alimentation est coupée. Son fonctionnement est similaire à celui d'un périphérique.
- 2 zones supplémentaires sont assimilables à de la mémoire :
  - les 32 registres de travail R0 à R31
  - les 64 registres I/O

Les registres sont des « cases mémoires » spécifiques qui peuvent etre spécialisés pour une utilisation particulière (PC, SP, ...)

Memoire programme

Un programme microcontrôleur est composée d'une suite d'instructions assembleurs (le langage du microcontrôleur est appelé son « jeu d'instructions ») qui sont exécutées les unes après les autres **.** Ces instructions sont « flashées » dans le microcontrôleur lors de la programmation de celui ci. Elles sont stockées dans la zone de mémoire appelée PROGRAM MEMORY en mémoire FLASH.

Les instructions du jeu d'instruction AVR ont une largeur de 16 ou 32 bits, la mémoire programme d'un ATMega 16 est de 16koctets = 8192 mots de 2 octets. Une instruction occupe 1 ou 2 mots de 2octets=16bits soit 16 ou 32 bits suivant les instructions.

Ces instructions sont donc stockées à des adresses allant de 0 à 8191 soit 0x0000 à 0x1FFF en hexadécimal. Ces adresses sont représentable par un nombre binaire de 13 bits (2^{13} = 8192).

La numération hexadécimale (base 16) contient 16 chiffres : 0 ...9 A B C D E F
Soit A=10, B=11 ...F =15 en décimal. En langage C, le préfixe **0x** indique que le nombre est écrit en hexadécimal (par exemple 0x86 n'est pas 86).

Conversion Hexadécimal --> Décimal

0x1FFF = **1***16^3 +**F***16^2+**F***16^1+**F** = 1*4096+15*256+15*16+15 = 8191
0x03FA=...

Conversion Décimal --> Hexadécimal

857/256 = **3**, il reste 857-3*256= 111
111/16= **6**, il reste 111-6*16 =15 = **F**

857=3*256 + **6***16+15 = **0x36F**

Conversion binaire <-->hexadécimal la conversion est très simple car 1 chiffre

hexadécimal = 4 chiffres binaires

0x03FA <-> 0 3 F A
0000 0011 1111 1010

Remarque : Pour les conversions décimal <-> binaire, il est très souvent plus simple de passer par une conversion intermédiaire en hexadécimal.

Le registre **PC = Program Counter** est un registre spécial qui stocke toujours l'adresse de la prochaine instruction à exécuter.

Pour l'ATMega16 qui dispose de 8192 $=2^{13}$ adresses pour stocker des instructions, seuls les 13 bits de poids faible de **PC** sont utilisés.

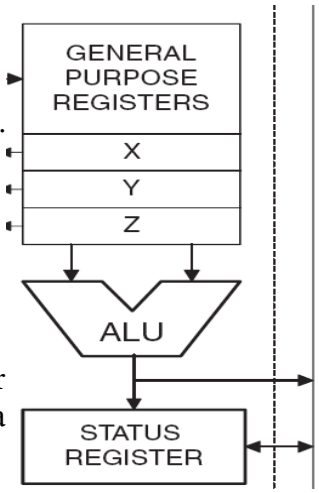
Lors de l'exécution d'une instruction : **PC** est incrémenté de 1 unité ou remplacé par l'adresse d'une nouvelle instruction (cas des sauts).

Registres génériques

Les microcontrôleurs AVR disposent de 32 registres génériques directement reliés a l'unité de calculs arithmétiques et logiques (ALU) en charge de l'exécution de ce type d'instructions(ADD, SUB, OR, AND, etc...).

Ces registres sont notées R0 à R31. L'ALU ne travaillent QUE sur ces 32 registres.

Exemples d'instructions :



ADD R21,R25
Additionne le contenu 8 bits du registre R25 au contenu de R21.

R25 est appelé **registre source** (Rs), R21 est le **registre de destination** (Rd)

Le mnémonique de ADD est **ADD Rd,Rs**

Il est a noté que le contenu de R25 n'a pas été modifié par l'instruction, le résultat est placé dans la destination R21, ce qui a détruit son contenu initial

ANDI R21,0x0F
Réalise un « ET logique » bit a bit entre 0x0F (0000 1111) et le contenu de R21.

ANDI = **AND Immédiat** car la valeur 0x0F est disponible immédiatement (l'ALU n'a pas à aller la prendre dans un registre)

Les registres IO

Les microcontrôleurs AVR disposent de 64 registres I/O, numérotés 0x00 à 0x3F. Ces registres permettent de :

- Contrôler les périphériques d'entrée/sortie,
- Contrôler certains modes de fonctionnement du processeur

1 registre particulier concerne le fonctionnement général du processeur.

Le **Status REGister SREG** est le registre de statut du processeur (n° 0x3F sur l'ATMega16):

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Il se compose de 8 bits dont les 6 derniers sont mis a jour en fonction du résultat d'exécution des instructions arithmétiques et logiques :

Bit 7 : **I**nterrupt : autorise globalement les interruptions
Bit 6 : **T**emp : permet le stockage temporaire d'un bit
Bit 5 : **H**alf carry : demie retenue utilisée dans certains calculs 4 bits
Bit 4 : **S**ign : Bit de signe indique si le résultat est positif ou négatif (S= N xor V)
Bit 3 : o**V**erflow : Bit de dépassement indique qu'un résultat a dépassé
Bit 2: **N**egative : Bit de dépassement indique un résultat Négatif
Bit 1 : **Z**ero : Bit de dépassement indique qu'un résultat a dépassé
Bit 0 : **C**arry : retenue

La mémoire SRAM et la PILE

La mémoire SRAM est l'espace mémoire qui contient les données. Sur un ATMega 16 sa taille est de 1024 octets.

Si on y ajoute les 32 registres généraux et 64 registres I/O, il y a donc 32+64+1024 = 1120=0x45F cases mémoires (de 8 bits) utilisables. 32+64 = 96 de ces emplacements mémoires ont une utilisation spécifique dédiée (registres généraux et registres IO).

Les 32 registres généraux R0-R31 occupent les adresses 0x00 à 0x1F
Les 64 registres I/O (N° 0x00à 0x3F) occupent les adresses 0x20 à 0x5F
Les 1024 octets de mémoire SRAM de l'ATmega16 occupent les adresses 0x60 à 0x45F

ATTENTION : Adresse Registre IO = N° registre IO + **0x20**

Un zone particulière de la SRAM est dénommée PILE. La pile contient les variables temporaires (variable locale en C) et des adresses. Ces variables et adresses sont em**PILÉ**es, les unes au dessus des autres.

Cette PILE commence en générale à l'adresse maximum 0x45F pour un ATMega16 et s'étend vers les adresses inférieures au fur et à mesure des besoins (instructions de gestion de la pile PUSH, POP).

Le Pointeur de pile (SP) ou STACK POINTER

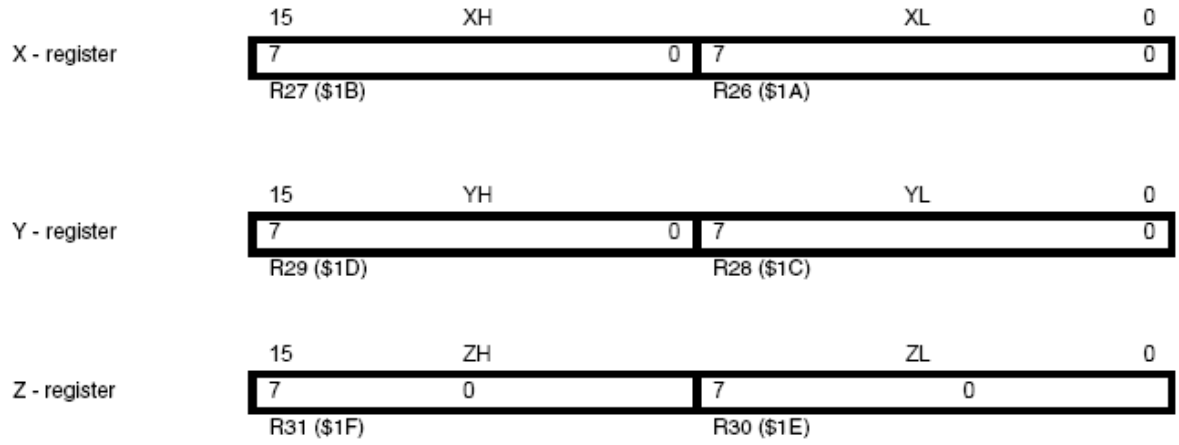
Par pointeur il faut comprendre registre spécifique prévu pour contenir une ADRESSE MEMOIRE.

Le pointeur de Pile (SP) est un pointeur 16 bits particulier. Il est composé de 2 registres SPH:SPL (n° 0x3E:0x3D des registres IO pour un ATMega16). Il pointe toujours sur le haut de la PILE, l'adresse mémoire pointée est toujours VIDE et prête à recevoir une donnée temporaire [1]

Les pointeurs X,Y,Z

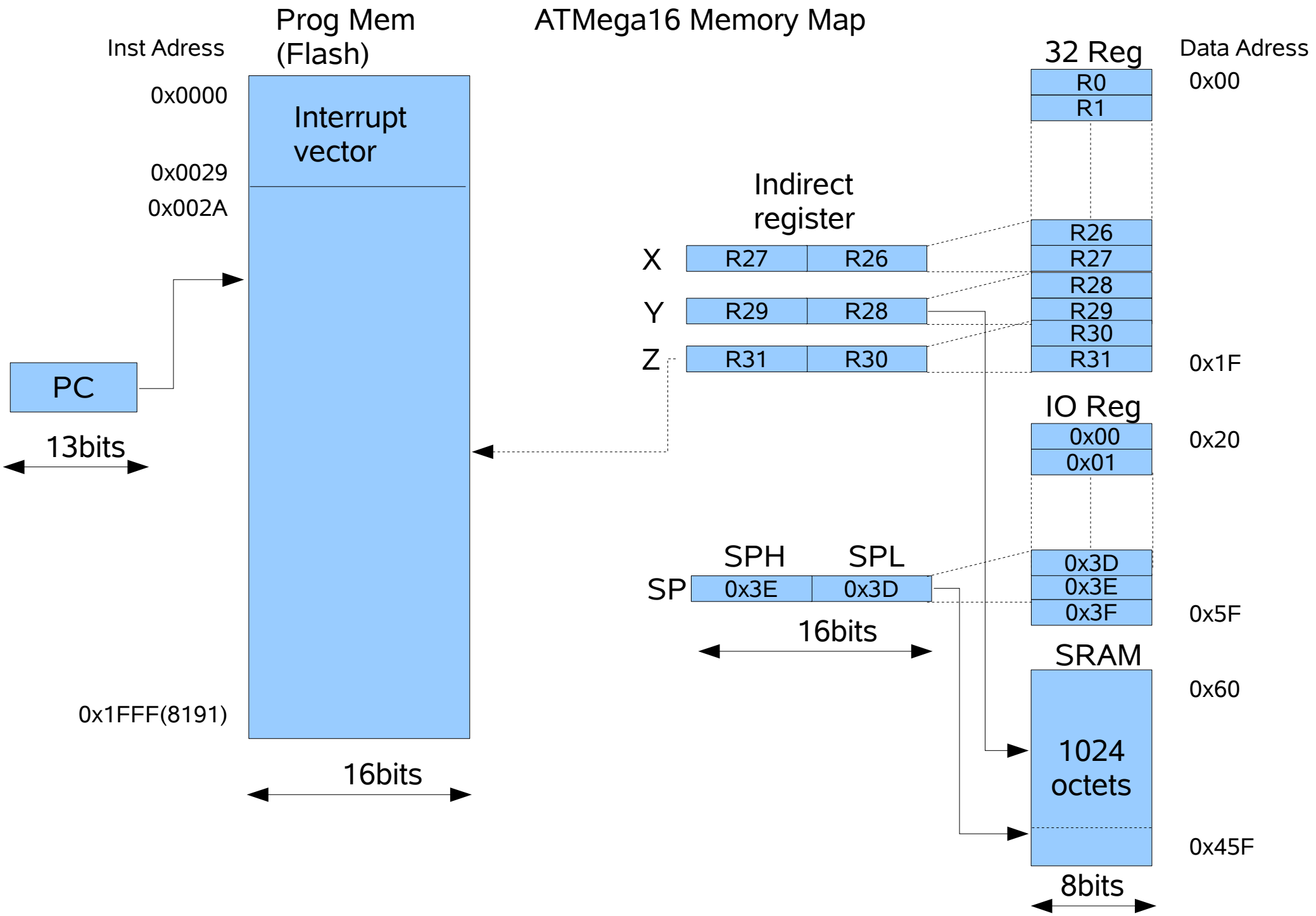
3 pointeurs 16 bits spécifiques notées **X, Y** et **Z** existent dans les processeurs AVR. Ils peuvent contenir une adresse de la mémoire de données (X,Y,Z) ou de la mémoire programme (Z).

Attention :ces 3 pointeurs utilisent les registres R26 à R31. Par exemple le registre X correspond à la concaténation de R27:R26. R27 contient les 8 bits de poids fort de X (**XH**igh), R26 les 8 bits de poids faible de X (**Xl**ow).



Ces 3 pointeurs sont utilisés pour l'adressage Indirect des données (voir mode d'adressage).

1 le STACK OVERFLOW étant l'exception


ATMega16 Memory Map
Prog Mem
(Flash)
Inst Adress
0x0000
Interrupt
vector
0x0029
0x002A
0x1FFF(8191)
16bits
PC
13bits
Indirect
register
X
Y
Z
R27
R26
R29
R28
R31
R30
32 Reg
Data Adress
0x00
R0
R1
R26
R27
R28
R29
R30
R31
0x1F
IO Reg
0x00
0x01
0x20
SPH
SPL
SP
0x3E
0x3D
16bits
0x3D
0x3E
0x3F
0x5F
SRAM
0x60
1024
octets
0x45F
8bits

Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	9
$3E ($5E)	SPH	–	–	–	–	–	SP10	SP9	SP8	12
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	12
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register								85
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	48, 69
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	70
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	85, 115, 133
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	86, 115, 133
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	250
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	180
$35 ($55)	MCUCR	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	32, 68
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	41, 69, 231
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	83
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								85
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register								30
	OCDR	On-Chip Debug Register								227
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	57,88,134,201,221
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	110
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	113
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								114
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								114
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								114
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								114
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								114
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								114
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								114
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								114
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	128
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)								130
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register								130
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	131
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	43
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]				167
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	166
$1F ($3F)	EEARH	–	–	–	–	–	–	–	EEAR8	19
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								19
$1D ($3D)	EEDR	EEPROM Data Register								19
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	19
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	66
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	66
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	66
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	66
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	66
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	66
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	67
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	67
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	67
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	67
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	67
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	67
$0F ($2F)	SPDR	SPI Data Register								142
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	142
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	140
$0C ($2C)	UDR	USART I/O Data Register								163
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	164
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	165
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte								167
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	202
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	217
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	219
$05 ($25)	ADCH	ADC Data Register High Byte								220
$04 ($24)	ADCL	ADC Data Register Low Byte								220
$03 ($23)	TWDR	Two-wire Serial Interface Data Register								182
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	182

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	181
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register								180

Notes:
1. When the OCDEN Fuse is unprogrammed, the OSCCAL Register is always accessed on this address. Refer to the debugger specific documentation for details on how to use the OCDR Register.
2. Refer to the USART description for details on how to access UBRRH and UCSRC.
3. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
4. Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.

5) Central Processing Unit

La CPU est le coeur du microcontrôleur. La CPU à pour rôle de
- charger les instructions depuis la mémoire programme vers le registre d'instructions
- décoder ces instructions en vue de les exécuter
- exécuter ces instructions. Pour cette exécution des signaux électriques (control line) de commande sont « envoyés » aux différents blocs constituant la CPU.

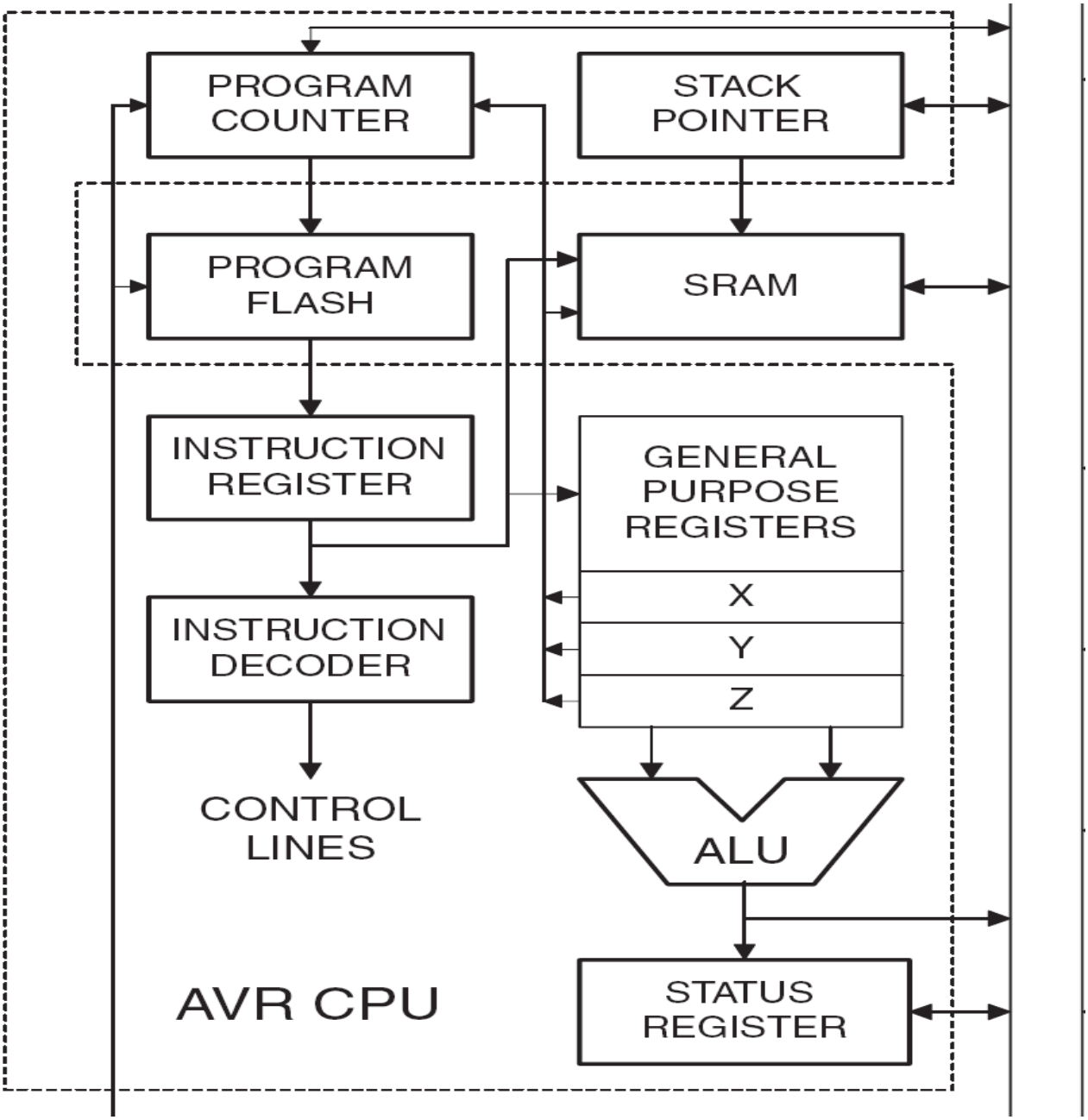
Le CPU se compose de :
- l'ALU : unité arithmétique et logique chargée d'éxecuter les instructions arithmétiques et logiques.
- 32 registres R0 R31 sur lequel l'ALU travaille
- 1 registre de statut SREG contenant certains indicateurs sur le résultat de l'instruction qui vient d'être exécutée par l'ALU
- 1 registre PC (Program Counter) indiquant l'adresse de la prochaine instruction à exécuter.
- Un unité logique de décodage et séquence ment des instructions qui contient 2 registres INSTRUCTION REGISTER (IR) et DECODE REGISTER (IR)
- le registre de PILE (SP) pointant sur le haut de la pile.

Si l'instruction à exécuter manipule des données, celles ci doivent être impérativement contenues dans un des 32 registres de travail de l'unité arithmétique et logique (ALU).

Pour séquencer ces instructions la CPU dispose d'un registre spécial : le PROGRAM COUNTER

Le registre PROGRAM COUNTER (PC) indique toujours l'adresse mémoire de la prochaine instruction à exécuter.

Quand les instructions sont exécutées les unes après les autres, PC s'incrémente de 1 instruction (1 ou 2 adresses suivant la taille de l'instruction) dès que l'instruction en cours a fini son exécution SAUF si l'instruction est un saut auquel cas le registre PC est modifié par cette instruction.



On peut classer les instructions en 5 types :

- les instructions arithmétiques et logiques : elles exécutent des « opérations » sur des données (une addition par exemple)
- les instructions de manipulations de bits
- les instructions d'accès mémoire : elles permettent de charger des données de la mémoire vers la CPU ou depuis la CPU vers la mémoire
- les instructions de saut et saut conditionnel : ces instructions permettent des ruptures de séquences
- des instructions de contrôle du CPU : mise en veille, arret....

Exemple de programme

Ce programme suivant resulte de la compilation du code C

```
unsigned char a=0x10,b=0x20,c;

void main void{
     c=a&b;
}
```

Adress inst	Instruction code	Instruction	
+00000055	91800060	LDS R24,0x0060	Load direct from data space
+00000055	91900062	LDS R25,0x0062	Load direct from data space
+00000057	2389	AND R24,R25	Logical AND
+00000058	93800061	STS 0x0061,R24	Store direct to data space
+0000005A	CFFF	RJMP PC+0xFFF+1	Relative jump

Remarques :
- il y a des instructions de 2 ou 4 octets, les adresses s'incrémentent en conséquence de 1 ou 2.
- On y retrouve 3 des 5 types d'instructions du processeur
  - LDS/STS accès mémoire
  - AND Opération logique entre 2 registres (traitée par l'ALU)
  - RJMP instruction de saut relatif (modifie le prog counter PC)
- Les instructions LDS contiennent l'adresse dans l' « instruction code » (0060 et 0061)

Analyse du programme : les données a, b et c sont stockées aux adresses 60,62 et 61.
PC = 00000055

6) Mode d'adressage et jeu d'instructions

Le jeu d'instruction des microcontroleurs AVR contient 5 types d'instructions
- arithmétiques et logiques
- manipulation de bits
- accès mémoires
- sauts et tests
- contrôle du processeur

Ces instructions sont codées sur 2 ou 4 octets.

On appelle OpCode (pour operation code) le code binaire (ou héxadécimal) de l'instruction. C'est cet « opcode » qui est décodé est exécuté par le CPU (exemple CFFF opcode d'un instruction RJMP).

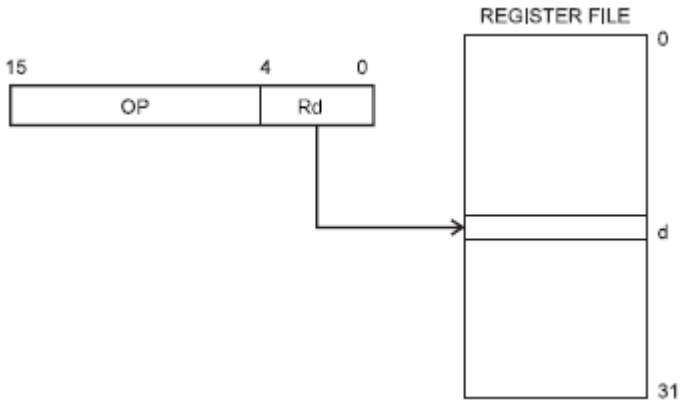
On appelle mnémonique le nom de l'instruction par exemple ADD, SUBI, MULS, LDS, RJMP
Le mnémonique n'est qu'une écriture plus humanisé de l'opcode.

6-1) Modes d'adressage

Les mode d'adressages désignent la manière dont une instruction connait l'adresse (ou le n° de registre) des données qu'elle doit manipuler.

Il y a 4 modes d'adressage principaux (et quelques variantes) :
- l 'adressage par registre(s),
- l'adressage immédiat,
- l'adressage direct
- l'adressage indirect

Adressage par registre unique



Le n^0 du registre (entre 0x00 et 0x1F) est indiqué dans opcode.

Illustration 1: Adressage par registre unique

Exemple INC R20 : incrémente le registre R20

L'opcode de cette instruction a une partie fixe et les 5bits **ddddd** représentent le numéro du registre destination. 20 = 0x14 = 1 0100

	Syntax:	Operands:
(i)	INC Rd	$0 \le d \le 31$

16-bit Opcode:

1001	010d	dddd	0011

L'opcode de **INC R20** est donc 1001 010**1 0100** 0011 soit 0x9543

Adressage par registre double

C'est le cas des instructions ALU a 2 opérandes .

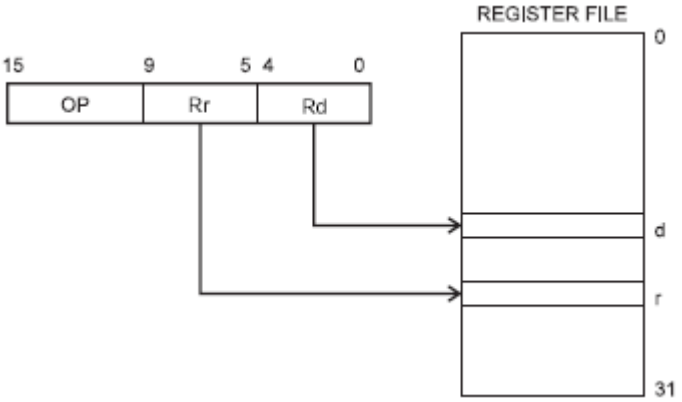


Illustration 2: Adressage par registre double

Exemple

AND R24,R25 : Et logique entre R24 R25

Syntax: **Operands:**
(i) AND Rd,Rr $0 \le d \le 31, 0 \le r \le 31$

16-bit Opcode:

0010	00rd	dddd	rrrr

	OP	0010	00			
Reg d=24 = 11000,	d =24			1	1000	
Reg r=25 = 11001	r =25			1		1001

L'opcode est 0010 00**11 0001 1001 = 0x2389**

Adressage Immédiat

La donnée est fournie « immédiatement », elle est contenue dans l'opcode

Exemple **ANDI R18,0x0F**

AND Immediate

Syntax: **Operands:**
(i) ANDI Rd,K $16 \le d \le 31, 0 \le K \le 255$

16-bit Opcode:

0111	KKKK	dddd	KKKK

Ici K=0x0F, la seconde opérande va être encodée à l'intérieur de l'OPCODE. Attention il y a une restriction sur le choix de Rd (de 16 à 31) afin de pouvoir coder ce numéro de registre sur 4 bits.

R18 = 0x02 le 3ème registre utilisable par cette instruction du fait de la restriction $16 \le d \le 31$
K = 0000 1111

l'OPCODE sera donc 0111 0000 0002 1111

Adressage Direct

L'adresse d'une donnée est donnée DIRECTEMENT dans l'OPCODE

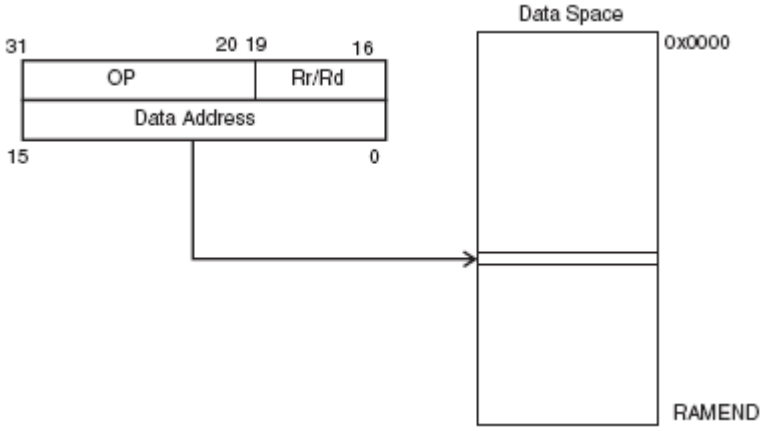


Illustration 3: Adressage direct en memoire donnée

Exemple LDS R24,0x0060
Load **D**irect from data **S**pace

0x0060 est l'adresse de la donnée en SRAM.
d=24 = 1 1000 le n° de registre

	Syntax:	Operands:
(i)	LDS Rd,k	0 ≤ d ≤ 31, 0 ≤ k ≤ 65535

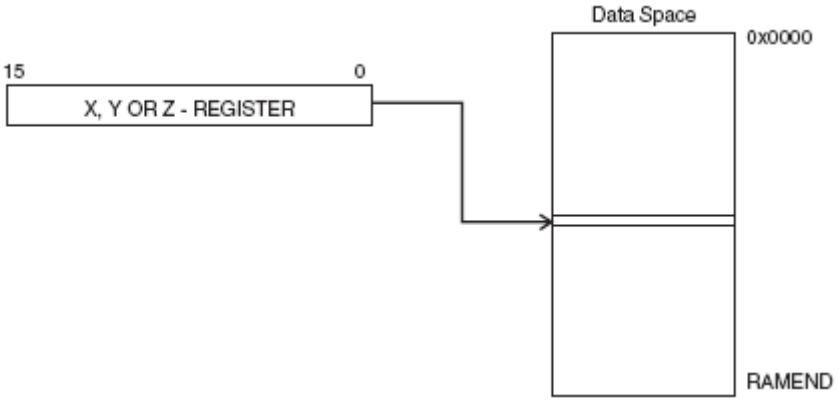
32-bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

l'opcode est 1001 000**1 1000** 0000
0000 0000 0110 0000 = 0x91800060

Adressage indirect par pointeur X,Y,Z

l'adresse de la données est contenu dans une des registre X,Y,Z



Exemple **LD R12,X**
Load Data Memory Indirect X

Charge la donnée dont l'adresse en mémoire SRAM est contenue dans X dans le registre R12

c'est le cas (i) du tableau opcode 1001 000**1 1000** 1100 = 0x918C

L'adressage indirect est très utilisé pour accéder aux tableaux (`char A[10]` en C par exemple) et dans ce cas il est très courant d'accéder successivement à tous les éléments du tableau (`A[i++]` ou `A[--i]`)

Operation:

(i)	Rd ← (X)	
(ii)	Rd ← (X)	X ← X + 1
(iii)	X ← X - 1	Rd ← (X)

	Syntax:	**Operands:**
(i)	LD Rd, X	0 ≤ d ≤ 31
(ii)	LD Rd, X+	0 ≤ d ≤ 31
(iii)	LD Rd, -X	0 ≤ d ≤ 31

16-bit Opcode:

(i)	1001	000d	dddd	1100
(ii)	1001	000d	dddd	1101
(iii)	1001	000d	dddd	1110

Illustration 4: LD Rd,X

L'adressage indirect des microcontrôleurs AVR supporte 2 variantes :

- la post incrémentation LD R12,X+
  le registre X est incrémenté d'une unité après l'accès à la valeur pointée par X

- la pré décrementation LD R12,-X
  le registre X est décrémenté AVANT l'accès à la valeur pointé par X

6.2) Jeu d'instruction

Le tableau du jeu d'instructions résume pour chaque instruction : son mnémonique, ses modes d'adressages, l'effet de l'opération sur les registres et PC, les drapeaux de SREG modifiés, et le nombre de cycles CPU nécessaire a l'exécution.

Instruction ALU :

On y retrouve tout le groupe d'instruction de « calcul » :
- Additions/soustractions : ADD,ADC,ADDW, INC, SUB, SUBI, SBC, SUBIW, DEC...
- Opération logiques : AND, ANDI, OR, ORI, EOR
- les multiplications MUL, MULS, FMUL, FMULS
- affectations particulière SET, CLR,

Instructions de manipulations au niveau bit.

Les instructions de rotations et décalage de registre (habituellement classées avec les instructions ALU) : LSL, LSR, ROR, ROL, ASR

Les instructions d'accès aux registres IO au niveau bit. Ces instructions permettent d'accéder bit a bit a ces registres. Seuls les registre IO de 0 a 0x1F (les 32 premiers donc) sont accessible en mode bit a bit. Cela inclus les registres gérant les ports d'entrée sortie A,B,C, D dont les numéros sont inférieur à 32.

Le groupe des instructions qui manipulent les bits du registre de statut SREG
- SEC (set Carry flag), CLC, SEN,CLN ....
  il y a un couple d'instruction (CLER/SET) pour chacun de ces bits.

Instructions d'accès mémoire

Accès à la **mémoire donnée**

- MOV pour copie entre registre
- LD pour chargement depuis la mémoire
- ST pour écriture en mémoire

Pour l'accès en lecture en mémoire programme par adressage indirect à l'aide du registre Z :

- STM pour charger le registre Z
- LDM pour lecture de la mémoire programme par adressage indirect par le registre Z

Il n'est pas possible d'écrire en mémoire programme.

Pour les accès aux registres IO

- IN et OUT **ATTENTION** **IN R2, 23** charge R2 avec le contenu du registre IO n° 23 (d'adresse SRAM 0x43 donc)

Pour acces à la pile

- PUSH, POP

Le fonctionnement de ces 2 instructions est très spécifique :

La pile est gérée par le pointeur de pile SP (Stack Pointer). Le pointeur de pile est initialisé à la fin de la mémoire (SP = 0x45F sur un ATmega16)

Dans l'exemple suivant on échange le contenu de 2 registre (R12 et R13) en passant par la pile. On suppose que SP = 0x45F (fin de la mémoire SRAM d'un ATMega16). Le code est le suivant :

PUSH R12	Le contenu de R12 est copié à l'adresse contenu dans SP (0x45F), R12 est copié sur le dessus de la pile. SP est décrementé(0x45E) et pointe toujours sur le haut de la pile
PUSH R13	Le contenu de R13 est copié à l'adresse contenu dans SP (0x45E), R13 est copié sur le dessus de la pile (au dessus de R12) (0x45D), SP est décrementé
POP R12	Le haut de la pile (SP-1) qui contenait une copie de R13 est copié dans R12, SP est incrémenté (le haut de la pile descends)
POP R13	Le haut de la pile (SP-1) qui contenait la copie de R12 est copié dans R13, SP est incrémenté (le haut de la pile descends)

Instructions de branchement et sauts

Dans un programme et à l'exception des instructions de branchement et saut, le **Program Counter PC** s'incrémente après chaque instruction. Cela a pour effet d'exécuter les instructions dans l'ordre et les unes après les autres. Les instructions de branchement sont les seules instructions qui permettent de changer cet ordre naturel d'exécution des instructions. Ces instructions de branchement sont générés par les structure de contrôle du C du type `if, for, while ?` Et les appels de fonctions.

- JUMP, RJUMP, IJUMP permettent de faire un saut à une adresse programme spécifiée
- CALL, RET : couple d'instruction pour les appels et retours de fonctions
- RETI : retour d'interruptions
- SKIP : Sbxx permet de sauter l'instruction suivante si le bit spécifié est à 1 ou 0

- BRANCH : Bxxx : saut (relatif) à l'adresse donnée si la condition est vérifiée

Instructions spéciales
- NOP : ne fait rien .... Mais son éxécution dure 1 cycle d'horloge
- SLEEP : endort le processeur (faible consommation)
- WDR : Rendort de chien de garde....
- BREAK : arrete le processeur (debug)

Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-Chip Debug Only	None	N/A