Chapitre 27 - Sous le capot de l'Uno, un moteur : l'ATmega328P

Structure du microcontrôleur

Pour bien utiliser un microcontrôleur, il faut connaître sa structure interne afin de bien comprendre ce qu'il peut réaliser. En effet, ce que peut faire un microcontrôleur dépend de la façon dont il est construit et tous les microcontrôleurs n'ont pas les mêmes possibilités. Celui qui équipe le module Arduino, l'ATmega328P, permet déjà de nombreuses choses pour un microcontrôleur 8 bits.

La figure 27.1 (tirée de la datasheet) montre un schéma synoptique de la structure de ce microcontrôleur ; nous allons la commenter afin de comprendre les différentes parties et ce qu'elles peuvent réaliser. La description que nous allons faire restera extrêmement simple.

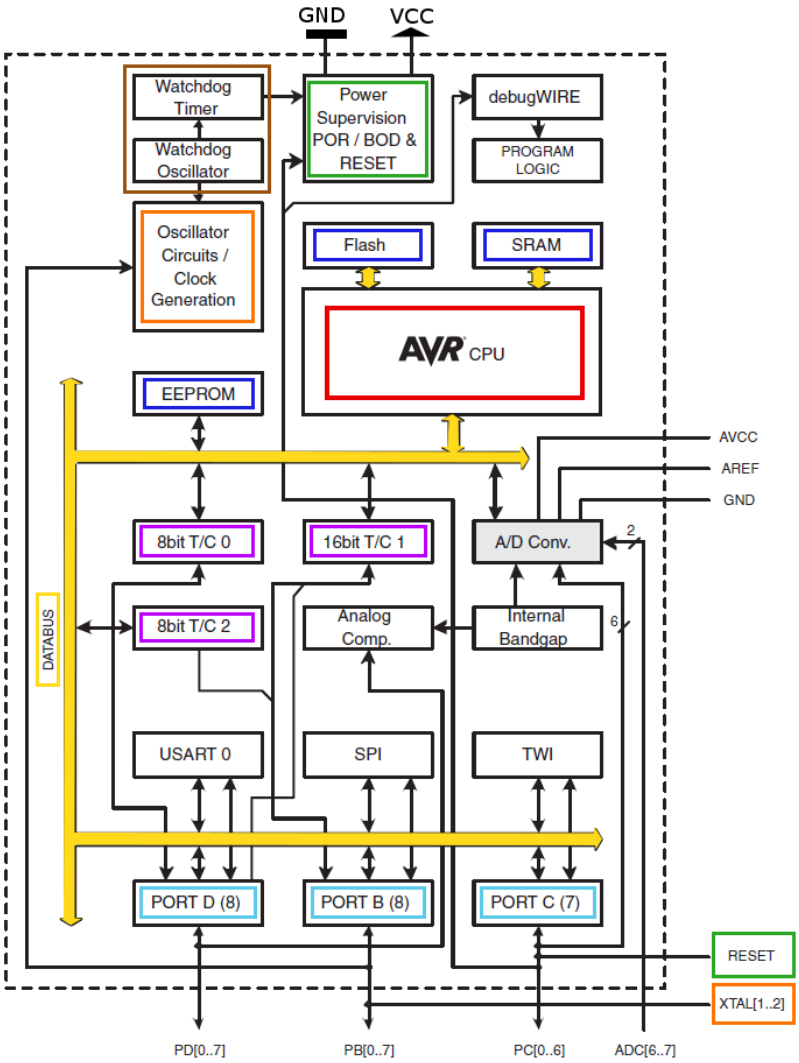

Figure 27. 1

Le cerveau du système est le CPU (en rouge) : Central Processing Unit ou microprocesseur. C'est cette unité qui réalise toutes les manipulations des données d'un programme, tous les calculs, toutes les opérations logiques ; elle contient l'ALU (Arithmetic and Logic Unit) dont nous avons déjà parlé. Le CPU est donc relié au bus de données représenté en jaune sur la figure (databus).

Pour fonctionner, le CPU a besoin d'être alimenté en courant. Une unité est spécialisée pour cela : la Power Supervision, représentée en vert. Elle-même est reliée à la source d'alimentation (VCC et GND). Le microcontrôleur ATmega328P a besoin d'une tension comprise entre 1,8 et 5,5 V pour fonctionner. Sur le module Arduino Uno, un régulateur de tension délivre du 5 V (plus ou moins 10%). L'unité d'alimentation se charge aussi d'effectuer le reset du microcontrôleur lors de la mise sous tension du circuit ; c'est ce qu'on appelle le POR (Power On Reset). Elle peut aussi effectuer un BOR (Brown-Out Reset) lorsque la tension d'alimentation tombe au-dessous d'un certain seuil. Enfin, cette unité effectue un reset lorsqu'un signal extérieur de reset est envoyé (le B/P Reset sur le module Arduino Uno) ; suivre le cheminement de ce signal sur le synoptique, vous le trouverez en bas à droite de la figure.

Le CPU manipule toutes les données d'un programme : il faut bien qu'elles soient stockées quelque part. Le microcontrôleur utilise pour cela de la mémoire, représentée en bleu foncé, et on voit qu'il en existe de trois sortes.

La mémoire Flash qui permet essentiellement de stocker les instructions d'un programme. Cette mémoire est analogue à celle des clés USB, elle se reprogramme facilement et l'information ne disparait pas lorsque le système est hors tension. Sur notre microcontrôleur, cette mémoire contient 32 kilo-octets de cases possibles pour stocker des données, mais en informatique, le kilo ne vaut pas 1000 mais 1024. Il y a donc 32 fois 1024 emplacements mémoires possibles, soit 32768 cases. En fait, toute la mémoire n'est pas disponible, car il y a le bootloader servant à la programmation du module Uno et qui représente à peu près 2 kilo-octets.

La mémoire SRAM (Static Random Access Memory) qui permet de stocker toutes les données pouvant varier au cours du programme (pour cette raison, on les appelle ces données « variables »). Elle est de 2 kilo-octets mais hélas, elle s'efface dès que le microcontrôleur est hors tension.

Pour résoudre ce problème, le microcontrôleur est doté de mémoire EEPROM (Electrical Erasable Programmable Read Only Memory) qui est une mémoire qu'on peut lire et écrire de façon électrique (contrairement à ce qu'indique l'acronyme qui n'a pas complètement évolué depuis les mémoires ROM programmées en usine) et qui retient l'information même en n'étant pas sous tension. Elle est de 1 kilo-octet.

Pour manipuler les différentes données d'un programme, le CPU a besoin d'une horloge (représentée en orange : Oscillator Circuits / Clock Generation) afin de fixer le rythme de travail. L'horloge est en quelque sorte le cœur du système puisqu'elle génère des pulsations électriques : il s'agit d'un circuit oscillant comme vous avez appris à en construire dans le chapitre 12. La vitesse des pulsations peut être déterminée avec un circuit RC (résistance et condensateur) mais ceci n'est pas très précis. Sur le module Arduino Uno, on fait appel à un cristal de quartz qui permet à l'horloge de générer des signaux précis de fréquence 16 MHz. Ce cristal est branché entre deux broches du microcontrôleur : suivez le cheminement du signal en partant d'en bas à droite de la figure. Les instructions élémentaires que le microcontrôleur connait, ne nécessitent qu'un simple cycle d'horloge pour être exécutées ; parfois il en faut plusieurs mais c'est assez rare, ce qui fait que ce microcontrôleur travaille rapidement, pratiquement 16 millions d'instructions par seconde.

Pour communiquer avec l'environnement extérieur, le CPU utilise des ports ; ils sont représentés en bleu clair et sont au nombre de trois, numérotés B, C et D. Nous en reparlerons. La communication CPU-PORT se fait par l'intermédiaire du bus de données (databus, en jaune) et aussi grâce aux registres de direction qui détermine si la ligne du port est une entrée ou bien une sortie. La figure 27.2 montre la correspondance entre les ports et les broches du module Uno.

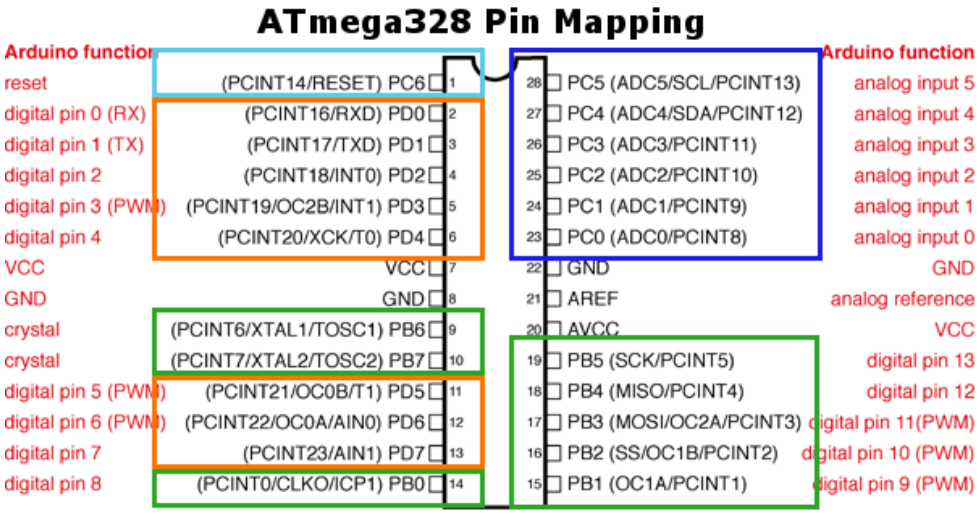

Figure 27. 2

Ce qui a été décrit jusqu'à présent pourrait être considéré comme le minimum nécessaire à un microcontrôleur, mais en général, il existe d'autres fonctions très intéressantes. Nous allons les décrire maintenant.

La première que nous pouvons remarquer et dont nous avons déjà un peu parlé, est un convertisseur analogique-numérique (A/D Converter) représenté sur fond grisé. Grâce à lui, nous disposons de 6 entrées analogiques (A0 à A5) sur notre module Uno, que nous pouvons lire pour connaitre la tension sur les broches. Ces entrées analogiques passent par le PORT C comme le montre la figure 27.2.

Le microcontrôleur dispose aussi de trois Timer/Counter, représentés en violet (T/C) et numérotés de 0 à 2. Un Timer/Counter est en quelque sorte un compteur de temps (un peu comme un minuteur en cuisine) ; on peut le régler sur une durée à respecter et lorsqu'on arrive à cette durée, le Timer/Counter se manifeste. À ce moment, le programmeur peut réaliser une action par l'intermédiaire d'un sous-programme, car c'est le moment de faire cela.

Deux de ces T/C comptent sur 8 bits (le T/C 0 et 2) et un compte sur 16 bits (le T/C 1) ce qui permet une durée de comptage plus grande. On peut faire beaucoup de choses avec les timers mais les manipuler demande une grande connaissance de la façon dont ils fonctionnent. Cela fait l'objet d'articles que j'ai écrits sur le site Locoduino. Les timers sont utilisés par le module Uno pour générer de la PWM (MLI Modulation par Largeur d'Impulsion en français) sur certaines broches.

Une autre particularité de ce microcontrôleur est la présence d'un chien de garde (Watchdog Timer) représenté en marron. Il s'agit d'un timer possédant sa propre horloge (Watchdog Oscillator) et dont la fonction est de reseter périodiquement le microcontrôleur, ce qui explique qu'il agit sur l'unité d'alimentation. On l'utilise donc pour éviter que le programme ne se fige : si c'était le cas, le chien de garde finirait par effectuer un reset du système et le programme redémarrerait au début. Si le programme fonctionne sans aucun problème, il faut juste penser à relancer ce timer chien de garde périodiquement avant qu'il ne déclenche le reset (sinon le programme n'ira jamais très loin). C'est donc au programmeur de déterminer si l'utilisation de ce chien de garde est utile ou non dans son application, car bien entendu, cette utilisation est optionnelle. Dans la majorité de nos applications, nous ne l'utiliserons pas, mais cette possibilité reste ouverte.

La description détaillée de chaque sous-ensemble dont nous avons parlé ici, nécessiterait un chapitre à part entière. Avant de clore ce chapitre, nous pouvons également dire quelques mots des cases (non colorées) figurant au-dessus des ports et dénommées USART, SPI et TWI.

USART : Universal Synchronous and Asynchronous serial Receiver and Transmitter, permet la communication série synchrone et asynchrone en émission ou en réception. C'est grâce à cette unité qu'on peut programmer le microcontrôleur via les broches Rx et Tx, celles-ci étant sur le PORT D.

SPI : Serial Peripheral Interface, permet aussi de générer des signaux pour communiquer en série comme MISO, MOSI et SCK, communication qui peut se faire entre un maître (module Uno par exemple) et des esclaves (périphériques). Ces signaux passent par le PORT B.

TWI : Two Wire Interface (ou I2C) est l'ensemble qui permet de générer les signaux SDA et SCL nécessaires à un bus I2C. Ces signaux passent par le PORT C.

À retenir sur le microcontrôleur :

- Le CPU qui est le cerveau du microcontrôleur ; il contient l'ALU.
- L'unité d'alimentation qui fournit le courant au microcontrôleur et gère les différentes formes de reset.
- La mémoire qui est de trois types : flash (programme), SRAM (données) et EEPROM (données).
- L'horloge qui peut travailler avec une cellule RC ou avec un quartz.
- Les différents PORT par lesquels le microcontrôleur communique avec l'environnement extérieur.
- Les timers (au nombre de trois) qui sont en quelque sorte des minuteurs.
- Le chien de garde et son utilité.
- Les autres unités du microcontrôleur telles que convertisseur A/D, l'USART, le SPI, le TWI.

Le CPU (Central Processing Unit)

Comme nous l'avons dit, le CPU est le cerveau du système : sa fonction principale est d'assurer l'exécution correcte du programme. Pour cela, le CPU doit être capable d'accéder aux mémoires, d'effectuer des calculs, de contrôler des périphériques et de gérer les interruptions dont nous avons parlé au chapitre 25. Pour que sa performance soit maximale, le microcontrôleur est construit selon une architecture Harvard, c'est-à-dire **avec des mémoires et des bus séparés pour les données et les programmes**. Pendant qu'il exécute une instruction, le CPU va chercher l'instruction suivante de la mémoire programme ; ce concept permet d'exécuter une instruction à chaque cycle d'horloge.

La figure 27.3 montre comment sont organisés les différents sous-ensembles permettant au CPU de fonctionner. Nous allons en faire un rapide survol car rentrer dans le détail dépasserait nettement le cadre de ce cours d'initiation.

Block Diagram of the AVR Architecture

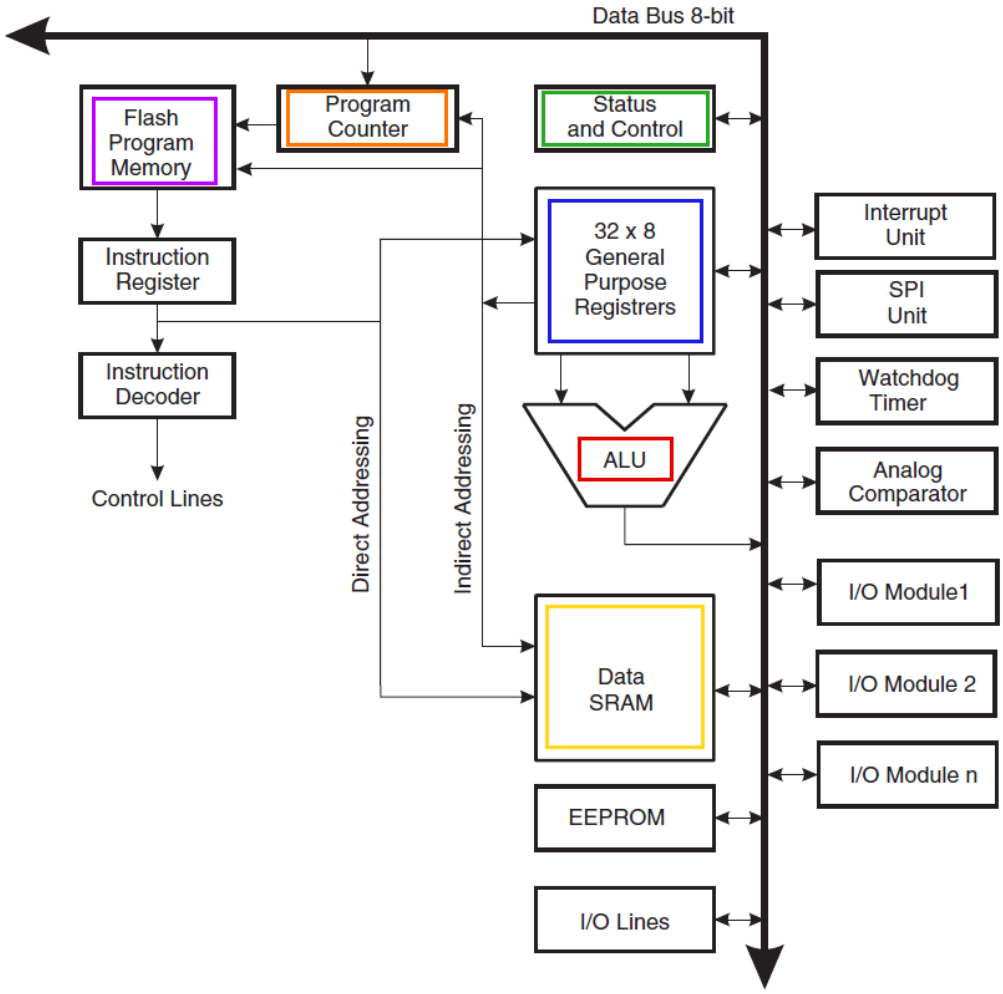

Figure 27. 3

La première chose que nous remarquons est l'ALU (Arithmetic and Logic Unit) encadrée en rouge. Cette unité effectue des opérations classées en trois catégories :

- Arithmétique
- Logique
- Manipulation de bits

L'ALU opère en connexion directe avec un espace mémoire de 32 octets encadrés en bleu, qu'on appelle des registres (General Purpose Registers ou encore registres à usage général). Par exemple, en un cycle d'horloge, une opération arithmétique entre deux de ces registres ou entre un registre et une donnée peut être effectuée. Ces 32 registres de travail ont des adresses allant de 0 à 31, soit en hexadécimal de 00 à 1F. La figure 27.4 montre l'organisation de ces registres dont les noms sont R0 à R31.

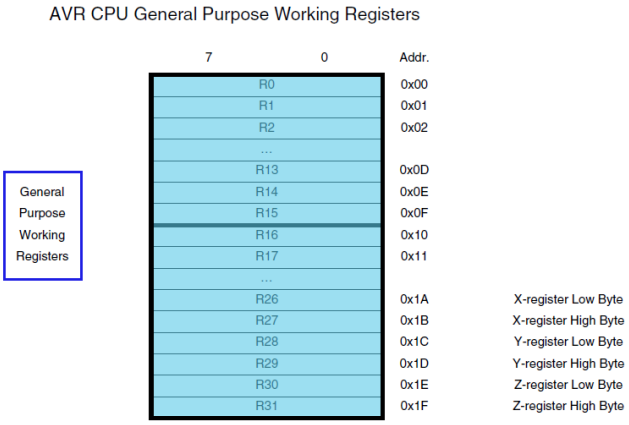

Figure 27. 4

La plupart des instructions opérant sur ces registres de travail ont des accès à l'ensemble des registres.

D'autres registres, donc d'autres emplacements mémoires bien spécifiques, permettent de contrôler le fonctionnement du microcontrôleur. Ils sont encadrés en vert. Un de ces registres s'appelle registre d'état (Status Register ou encore SREG) et contient des informations sur la dernière opération arithmétique exécutée. Par exemple, le bit 0 indique si le résultat d'une opération génère une retenue ; on l'appelle bit de Carry (retenue en anglais). Le bit 1 indique si le résultat d'une opération est zéro. Chacun des bits du registre d'état peut être lu ou écrit par le programmeur pour les besoins du programme à réaliser. Par exemple, on peut écrire dans le bit 7 pour autoriser ou non les interruptions. Le registre d'état est automatiquement mis à jour après chaque opération de l'ALU.

De nombreux registres permettent de faire fonctionner le microcontrôleur et de connaître en permanence son état. Par exemple, de nombreux registres contrôlent la façon dont opèrent les timers. Tous ces registres occupent des emplacements de la mémoire de données (Data SRAM) sans toutefois occuper l'espace mémoire SRAM à la disposition de l'utilisateur. La Data SRAM est encadrée en jaune et est organisée comme le montre la figure 27.5. On voit que tous les registres occupent le début de l'espace de données, avec des adresses allant de 0 à 255 (en hexadécimal, de 00 à FF). On trouve les 32 registres de travail à usage général dont on a parlé, puis 64 registres standard d'entrée-sortie et enfin 160 registres étendus d'entrée-sortie, ce qui fait au total 256 emplacements mémoires pour des registres (tous ne sont pas utilisés). La mémoire SRAM à disposition de l'utilisateur comme à l'adresse 256 (en hexadécimal 100) et a une taille de 2048 octets sur le microcontrôleur ATmega328P (dernière adresse en hexadécimal 8FF soit 2303 (255 + 2048)).

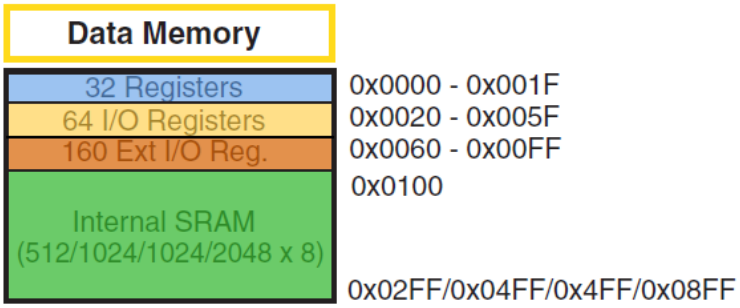

Figure 27. 5

Maintenant que nous avons passé en revue la mémoire de données, voyons la mémoire de programme et la façon dont le microcontrôleur exécute le programme. La mémoire de programme est encadrée

en violet sur la figure 27.3 et comme on l'a dit, elle est distincte de la mémoire de données. C'est dans cette mémoire que le microcontrôleur va chercher les instructions à exécuter. Celui-ci est bien incapable de comprendre un ordre comme digitalRead() ; tout ce que le microcontrôleur sait exécuter, ce sont des instructions élémentaires très simples. La figure 27.6 montre le début du jeu d'instruction de notre microcontrôleur. Par exemple, la première ligne est l'addition (mnémonique ADD en colonne 1) de deux registres de travail (colonne 2 : Rd et Rr) ; l'opération (colonne 4) indique que le résultat est mis dans le registre Rd ; on voit aussi (colonne 5) que cela met à jour les bits Z, C, N, V et H du registre d'état et que cette instruction nécessite un cycle d'horloge (colonne 6).

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 ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – 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 • (0xFF - 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

Figure 27. 6

S'il fallait connaître toutes les instructions possibles et tous les mnémoniques, la programmation du microcontrôleur serait fastidieuse (c'était le cas auparavant) ; heureusement, grâce à la solution Arduino et son langage de haut niveau, c'est devenu plus simple. Néanmoins, le travail de l'IDE lors de la phase de compilation, c'est de traduire une instruction comme digitalRead() en une suite d'instructions élémentaires de manière à effectuer la tâche demandée.

Le microcontrôleur exécute donc le programme en prenant les instructions les unes à la suite des autres pour les exécuter, un peu comme on peut suivre une recette de cuisine. Il a besoin pour cela d'un compteur de programme (Program Counter) encadré en orange sur la figure 27.3, pour repérer où il en est du déroulement du programme. Chaque fois qu'une instruction est exécutée, le compteur de programme augmente d'une unité pour pointer sur l'instruction suivante. Parfois, le programme peut appeler un sous-programme ; il doit donc interrompre son exécution pour aller exécuter les instructions du sous-programme, puis lorsque c'est terminé, revenir où il en était dans le programme. Ceci nécessite de sauvegarder le compteur de programme dans un espace mémoire qu'on appelle une pile, et comme il faut bien repérer où commence la pile, un registre (composé de deux octets puisqu'on stocke une adresse mémoire) est utilisé pour cela (Stack Pointer ou pointeur de pile). Le pointeur de pile est donc constitué de deux octets dans l'espace des registres d'entrée-sortie. En fait, la pile sert à stocker des données temporaires, des variables locales, et des adresses de retour en cas d'interruption ou d'appels de sous-programme. Au fur et à mesure qu'on empile des données, la pile grossit dans la mémoire depuis les plus hautes adresses vers les plus basses et le pointeur de pile pointe toujours le sommet de la pile (dernière donnée ajoutée). Le fait de sauvegarder une donnée dans la pile (Stack Push) décrémente le pointeur de pile d'une unité et le fait de la récupérer (Pop) l'incrémente d'une unité. Pour comprendre le concept de pile, on peut imaginer une pile d'assiettes ; quand on en rajoute une, c'est en haut de la pile qu'on la pose et quand on en prend une, c'est celle la plus en haut qu'on

prend ; la hauteur de la pile d'assiettes varie donc. La pile doit être définie dans la mémoire de données par le programme avant que des sous-programmes soient exécutées ou des interruptions autorisées (l'IDE d'Arduino s'en charge pour nous).

Grâce au compteur de programme, les instructions sont donc exécutées dans l'ordre qui convient. Pour gagner du temps, pendant l'exécution d'une instruction, le CPU va déjà extraire l'instruction suivante comme le montre la figure 27.7. Celle-ci est placée dans le registre d'instruction pour être ensuite décodée par le décodeur d'instruction (voir figure 27.3).

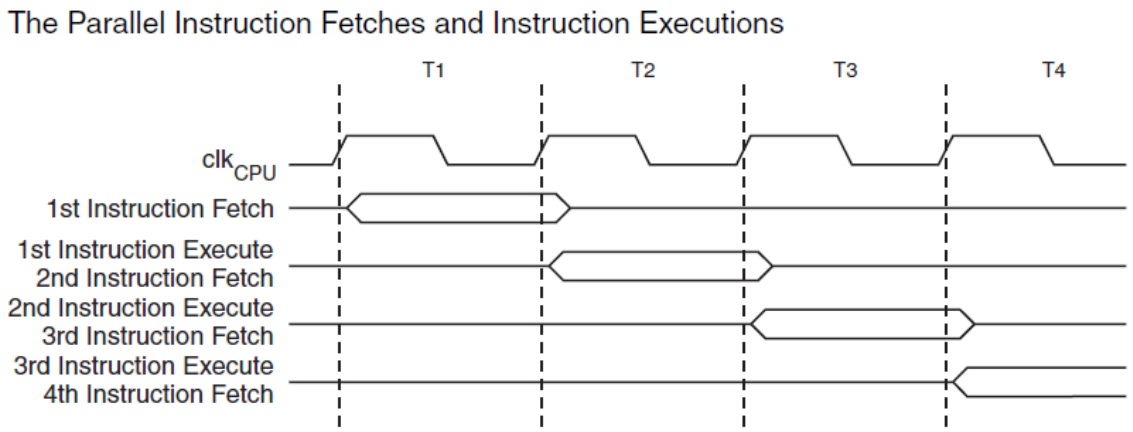

Figure 27. 7

De même, durant un cycle d'horloge, l'ALU réalise une opération sur deux registres et place le résultat dans le registre de destination, comme le montre la figure 27.8.

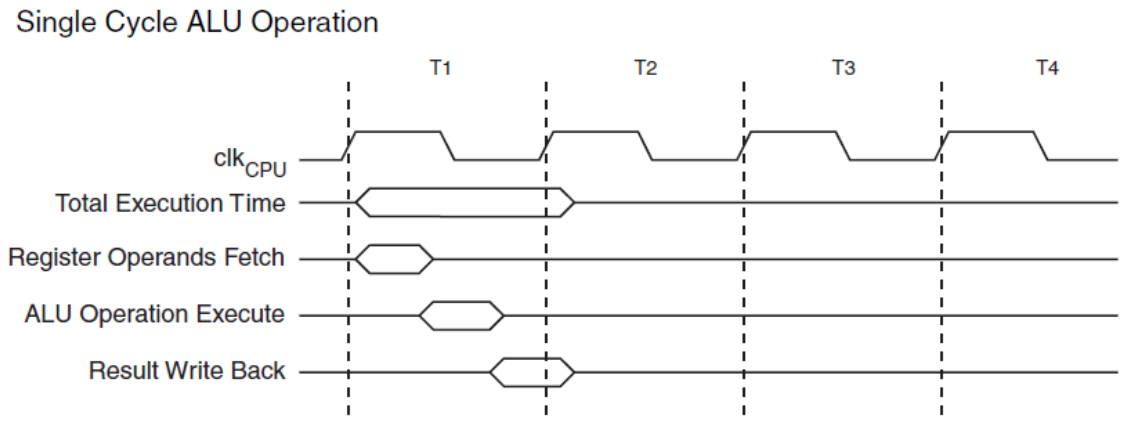

Figure 27. 8

La mémoire de programme est constituée d'une grande section réservée à l'application (le programme que vous avez écrit) et une petite section est réservée au Boot Loader qui permet de programmer la mémoire programme, ce que montre la figure 27.9. Les instructions élémentaires que comprend le microcontrôleur tiennent sur 16 bits ou 32 bits ; la mémoire flash de programme est donc organisée en fait selon 16 kilos de 16 bits (3FFF sur la figure 27.9), et le compteur de programme a 14 bits de large pour adresser ces 16 k d'emplacements de 16 bits. L'endurance de la mémoire flash est d'au moins 10000 cycles d'écriture-effacement, ce qui peut sembler assez peu mais on ne change pas les applications si souvent (la mémoire EEPROM supporte quant à elle 100000 cycles d'écriture-effacement).

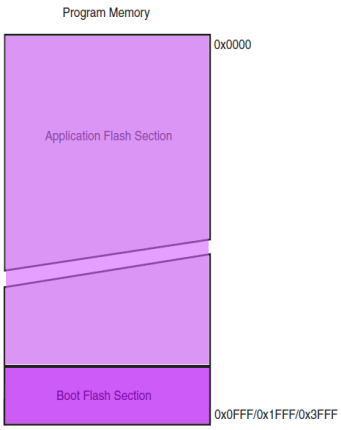

Figure 27. 9

Sachant que l'horloge du microcontrôleur peut monter jusqu'à 20 MHz, celui-ci peut exécuter quasiment 20 Mips (millions d'instruction par seconde) puisque la plupart des instructions ne nécessitent qu'un seul cycle d'horloge pour être exécutées. Sur le module Uno, l'horloge est réglée à 16 MHz par un cristal de quartz.

Le CPU est aussi capable de gérer d'autres unités comme les ports d'entrée-sortie, ou bien la mémoire EEPROM ou encore les timers. Pour faire tout cela, le CPU utilise des registres et chaque unité a ses propres registres. Nous reparlerons de certains registres, par exemple ceux qui gèrent les ports d'entrée-sortie et qui sont registre de direction DDR, registre d'entrée PIN et registre de sortie PORT, pour chacun des trois PORT B, C ou D. La figure 27.10 montre la localisation de ces registres. Pour programmer avec efficacité un microcontrôleur, il est nécessaire de bien connaître les registres et comment ils fonctionnent, et comme ils sont nombreux, cela demande du temps…

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–	
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–	
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–	
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–	
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	Reserved	–	–	–	–	–	–	–	–	
0x0D (0x2D)	Reserved	–	–	–	–	–	–	–	–	
0x0C (0x2C)	Reserved	–	–	–	–	–	–	–	–	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	93
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	93
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	93
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	92
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	92
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	92
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	92
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	92
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	92
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–	

ATmega328P

Figure 27. 10

NB : toutes les figures de ce chapitre ont été extraites de la datasheet de l'ATmega328P d'Atmel.

<u>À retenir sur le CPU</u> :

- Le CPU est le cerveau du système : sa fonction principale est d'assurer l'exécution correcte du programme.
- Le CPU doit être capable d'accéder aux mémoires, d'effectuer des calculs, de contrôler des périphériques et de gérer les interruptions.

- Mémoire de programme et mémoire de données sont distinctes avec des bus distincts (architecture Harvard).
- L'ALU est capable de réaliser des opérations arithmétiques, logiques et des manipulations de bits.
- Il existe 32 registres de travail à usage général, 64 registres standards et 160 registres étendus.
- La mémoire de données est constituée des emplacements mémoire de ces registres et des emplacements mémoire RAM mis à disposition de l'utilisateur.
- La mémoire de programme contient les instructions élémentaires que peut comprendre le microcontrôleur.
- Pendant qu'une instruction est exécutée, le CPU extrait l'instruction suivante, ce qui permet de gagner en rapidité d'exécution.
- Pour se repérer dans le déroulement du programme, un compteur de programme est nécessaire.
- Sa valeur est sauvegardée dans une pile (emplacement mémoire) en cas d'appels de sous-programme ou d'interruptions.
- Le pointeur de pile permet de gérer la pile dont la dimension varie au fur et à mesure que des valeurs sont sauvegardées ou récupérées.
- La mémoire de programme est divisée en deux sections, une pour l'application, une autre pour le Boot Loader qui permet de programmer le microcontrôleur.
- Les instructions élémentaires tiennent généralement sur 16 bits, parfois 32, donc la mémoire de programme est de 16 kilo-emplacements de16 bits (de 0000 à 3FFF).
- La mémoire flash contenant le programme supporte 10000 cycles d'écriture-effacement.
- Le microcontrôleur est capable d'exécuter 20 millions d'instructions par seconde.
- Le CPU utilise des registres pour gérer les différentes unités comme les ports d'entrée-sortie, les timers, la mémoire EEPROM, etc.