|  |  |  |
| --- | --- | --- |
| **Logo_Lycée** | **Langage machine et assembleur** | **logo%20ac%20orl%E9ans%20toursDescription : Description : Description : pemDescription : Description : Description : pem** |
| **TP**  **Synthèse** | **Date :** \_\_\_\_\_\_\_ **Nom :** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Prénom**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**1. Introduction**

Les programmeurs travaillent principalement avec des langages de haut niveau tels que le C, le JavaScript, le python, etc. Mais le **code source** de ces langages n’est pas directement exécutable par les processeurs : il doit être « transformés » en **code machine**.

**2. Langage machine**

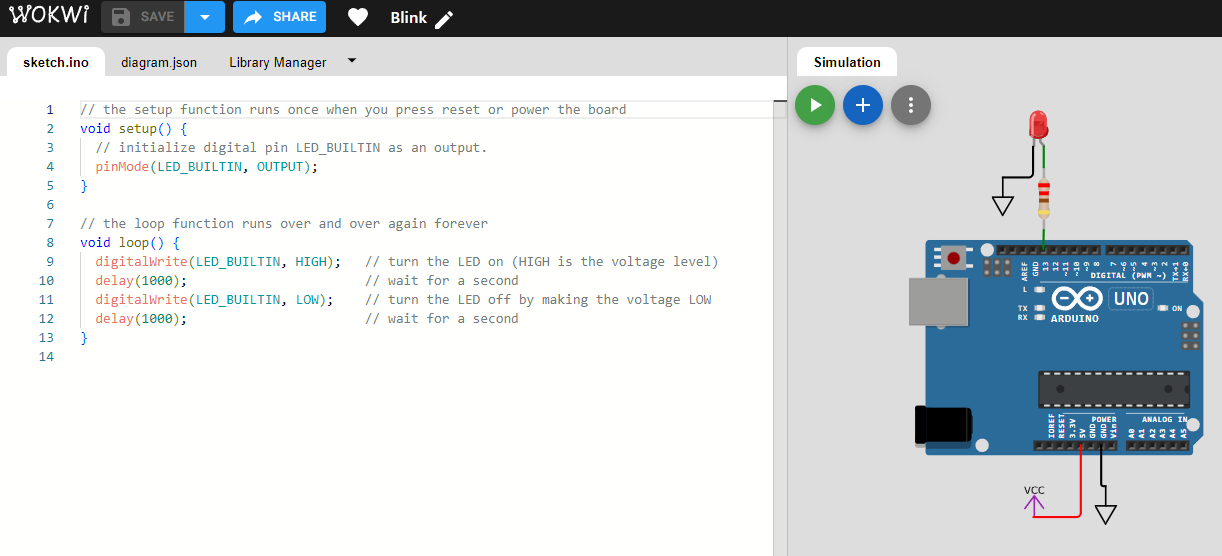
Définition : *« le* ***langage machine****, ou* ***code machine****, est la suite des* ***valeurs binaires*** *qui est interprétée par le* [*processeur*](https://fr.wikipedia.org/wiki/Processeur) *d'un ordinateur exécutant un programme informatique. C'est le* ***langage natif*** *d'un processeur, c'est-à-dire le seul qu'il puisse traiter. Il est composé d'*[*instructions*](https://fr.wikipedia.org/wiki/Instruction_informatique)*et de données à traiter codées en binaire. »* [Wikipédia](https://fr.wikipedia.org/wiki/Langage_machine)

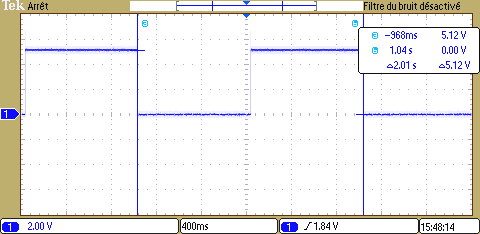
**2.1 Présentation de l’étude de cas : Hello World**

|  |  |
| --- | --- |
|  | La découverte d’un langage de programmation se fait souvent en écrivant un programme qui affiche un simple « *Hello World* » dans le but de faire la démonstration rapide de son exécution sans erreur. Cette pratique est attribuée à un des pionniers de l’informatique, [**Brian Kernighan**](https://fr.wikipedia.org/wiki/Brian_Kernighan) co-auteur du premier livre sur le langage C.  Dans les systèmes embarqués, le texte peut être remplacé par une simple **LED**. |

*Le programme «***Blink »**

La figure ci-dessous représente un montage permettant de faire clignoter une LED. Le code source du programme « Blink » est écrit avec la syntaxe Arduino (langage C++). [[Wokwi](https://wokwi.com/projects/405399027165376513)]





ATMEGA328P

Pour être exécuté par le **CPU** du microcontrôleur **ATMEGA328P**, le code source est **transformé** en code machine à l’aide d’un **compilateur**. Les valeurs binaires du code objet issu de la compilation sont stockées dans les mémoires (**flash, EEPROM)**.

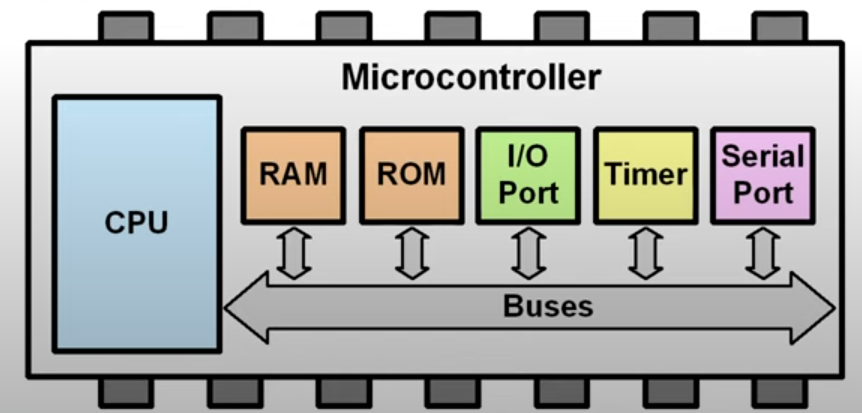
**2.2 Représentation fonctionnelle simplifiée du microcontrôleur**

*« Un microcontrôleur (en notation abrégée* ***µc****, ou uc ou encore MCU en anglais) est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires (mémoire morte et mémoire vive), unités périphériques et interfaces d'entrées-sorties. »* [*Wikipédia*](https://fr.wikipedia.org/wiki/Microcontr%C3%B4leur#:~:text=Un%20microcontroleur%20(en%20notation%20abr%C3%A9g%C3%A9e,interfaces%20d'entr%C3%A9es%2Dsorties.)

**Notes** : le sigle ROM (Read Only Memory) est utilisé ci-dessus pour représenter les mémoires qui conservent leur contenu hors tension (telles que la flash et l’EEPROM).

1ko = 1024 octets

1 octet = 8bits



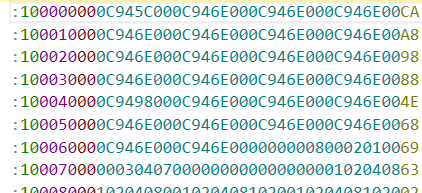
Exemple : ATMEGA328P - processeur : RISC 8bits (20MHz) - Mémoire programme flash : 32ko (16K x 16) – RAM : 2ko – EEPROM : 1ko

***Q1.*** ***Identifiez*** *les fonctions du modèle de* ***von Neumann*** *sur le schéma du microcontrôleur.*

**2.3 Compilation d’un code source**

|  |  |
| --- | --- |
|  | Le code machine correspondant au code source (fichier *Blink.cpp*) peut être obtenu en utilisant un environnement de programmation tel que l’IDE **Microchip Studio**. Ce codemachine, issu de la **compilation** du code source, est disponible dans un fichier dont l’extension est : [**.hex**](https://fr.wikipedia.org/wiki/HEX_(Intel)). |

*Exemple : e*xtrait du fichier *Blink.hex*issu de la compilation du code source situé dans *Blink.cpp*.



|  |  |
| --- | --- |
| Toutes les valeurs contenues dans un fichier .hex ne sont pas transférées dans la mémoire du microcontrôleur. Certaines sont utilisées pour contrôler le transfert entre l’ordinateur et le microcontrôleur. |  |

*Exemple* : les deux premières et les deux dernières lignes du fichier *Blink.hex* ont été réécrites ci-dessous.

*:10 0000* 00 **0C 94 5C 00 0C 94 6E 00 0C 94 6E 00 0C 94 6E 00** *CA*

*:10 0010* 00 **0C 94 6E 00 0C 94 6E 00 0C 94 6E 00 0C 94 6E 00** *A8*

…………

*:**0A 0420 00* ***1F 91 08 95 08 95 F8 94 FF CF*** *8E*

*:00000001FF*

Le code machine est représenté en caractères gras. Il est transféré dans la mémoire flash du microcontrôleur lors de sa programmation. Les autres valeurs sont utilisées pour le contrôle du transfert.

*Consultez la ressource* *[https://urlr.me/nTsRw](https://urlr.me/nTsRw ) pour répondre aux questions Q2, Q3 et Q4.*

*Chaque ligne du fichier Blink.hex commence par* ***:10****.*

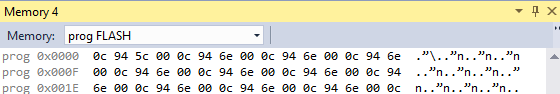
***Q2.****À quoi correspond cette valeur ? À quoi correspondent les deux octets qui suivent (Exemple : 0000, 0010, etc) ?* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Q3.** Combien d’octets contient le programme Blink ? La mémoire flash de l’ATMEGA328P est-elle suffisante ? Pourquoi ?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Exemple de résultat obtenu en flash*

La figure suivante représente la mémoire flash (partielle) d’un ATMEGA328P après le chargement du programme contenu dans le fichier *Blink.hex*.



***Q4****. Utilisez la calculatrice Windows en mode programmeur pour* *calculer le* *cheksum de la suite des valeurs hexadécimales :*

*0A 04 20 00 1F 91 08 95 08 95 F8 94 FF CF. Quel(s) réglage(s) avez-vous appliqué à la calculatrice ?*

\_cheksum = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3. Le langage de bas niveau – L’assembleur**

Bien que majoritairement utilisés, les langages de haut niveau ne permettent pas de savoir ce que fait la machine ni de la contrôler de façon précise et optimisée. Aussi un autre langage, compréhensible par l’être humain, mais très proche de la machine est utilisé : l’**assembleur**.

**3.1 L’outil logiciel désassembleur**

|  |  |
| --- | --- |
|  | L’IDE **Microchip Studio** offre la possibilité d’obtenir le code source assembleur du programme Blink à l’aide d’un outil logiciel appelé **désassembleur**. |

*Exemple*

Désassemblage de l’instruction ***pinMode(LED\_BUILTIN, OUTPUT);*** *par* ***Microchip Studio***

Note : La mémoire est organisée en mots de 16 bits

35: void setup() {

36: // initialize digital pin LED\_BUILTIN as an output.

37: pinMode(LED\_BUILTIN, OUTPUT);

00000070 61 e0 LDI R22,0x01 Load immediate

00000071 8d e0 LDI R24,0x0D Load immediate

00000072 0c 94 a7 01 JMP 0x000001A7 Jump

…

Le code machine de cette instruction dans le fichier .hex commence à la position E016.



Cet exemple fait apparaître la représentation d’un code source assembleur organisé en champs : **opcode et commentaires.** Cette organisation doit être respectée lors de l’écriture d’un programme en **assembleur** pour créer un code objet.

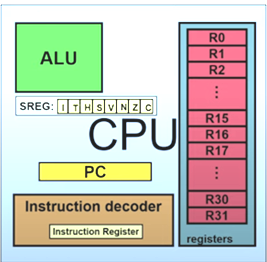
**Code machine en mémoire Assembleur (code source)**

*Adresse* ***16-bit opcode*** *instruction opérandes Commentaires*

*000000E0* **61 e0** ***LDI*** R22,0x01 **L**oa**D** **I**mmediate (0x = base 16)

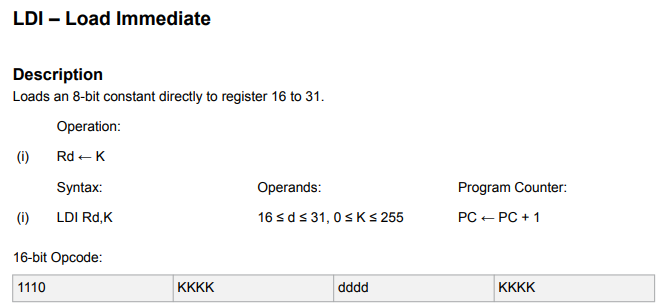
L’instruction **LDI R22, 0x01** signifie que le processeur place la valeur 1 dans son registre R22. (Il en possède 32)

La relation entre le code machine et l’assembleur est donné dans la documentation du composant.



CPU de l’ATMEGA328P

Exemple extrait de [AVR Instruction Set Manual](https://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf)



**Un programmeur averti comprendra ce que fait *LDI R22,0x01* (il saura même l’écrire), mais qu’en est-il du décodeur d’instruction du CPU ?**

L’instruction doit être codée selon la documentation ci-dessus. C’est le travail du logiciel assembleur mais dans certains cas cela peut être fait par le programmeur.

En observant l’extrait de code ci-dessus et la documentation, on en déduit que **LDI** est codé par **e**16 = **1110**2.

On retrouve facilement la valeur de K dans 16-bit Opcode en inversant l’ordre des octets afin qu’il corresponde à la documentation

e**061**16 = 1110 **0000 0110 0001**2 d’ou

|  |  |
| --- | --- |
| **Adresse** | **Opcode** |
| 00E0 | 61 E0 |
|  |  |

Mémoire flash

- KKKK = 0000 (quartet de poids fort de K) donc **K** = 000000012 = **0x01**

- dddd = 01102 = 610 de **d = 6** on peut déduire que le registre

- KKKK = 0001(quartet de poids faible de K) concerné est le sixième après R16 soit **R22**

Conclusion : le **décodeur d’instruction,** recevant la valeur **e061**16 que le CPU est allé chercher à la position **E016** dans la mémoire, comprendra qu’il doit placer la valeur 1 dans le registre R22.

**Q5**. Identifiez l’instruction assembleur correspondant au code 21 E4 en mémoire.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3.2 Réécrire « Hello World ! » en assembleur**

|  |  |
| --- | --- |
|  | L’IDE **Microchip Studio** offre également la possibilité de programmer **DIRECTEMENT** l’ATMEGA328P en **assembleur**. |

* **Blink.asm**

Faire clignoter une LED avec un ATMEGA328P se fait également avec le code ci-dessous écrit directement en assembleur. L’IDE **Microchip Studio** nomme le fichier source avec l’extension **.asm**.

**DELAY**

r27r26 ← 1000

r25r24 ← 4000

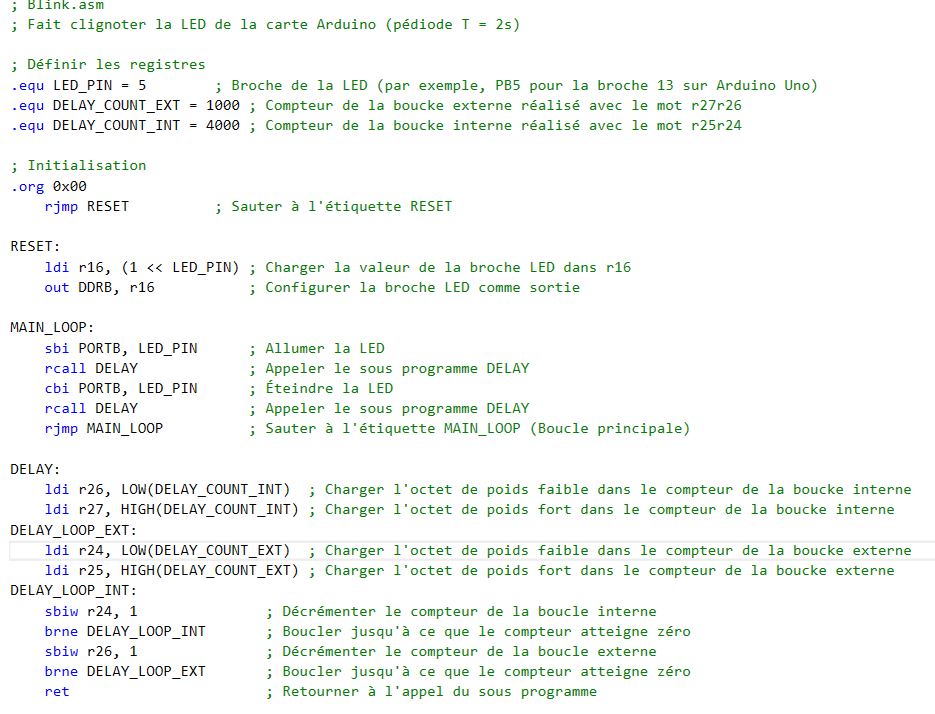
r25r24 ← r25r24 - 1

r25r24 = 0

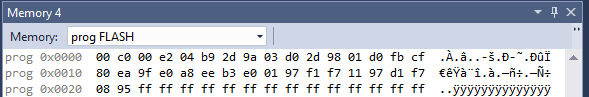
r27r26 ← r25r24 - 1

r27r26 = 0

**fin**



*Code machine du fichier Blink.asm dans la mémoire flash*



**Q6**. Comparez l’empreinte mémoire du programme *Blink.ino* avec celle de *Blink.asm* exprimé en pourcentage de la mémoire total. Conclusion.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* **Écriture de Blink.asm dans l’environnement Arduino et mise en œuvre sur une Arduino UNO**



On introduit du code assembleur dans l’IDE Arduino avec la fonction asm(). Le programme ci-dessus a été adapté cet IDE. Ce programme est disponible dans le fichier ***Blinkasm.ino*** (Voir prof pour le télécharger)

Comme dans le programme ci-dessus, la temporisation (Delay) est réalisée avec deux boucles imbriquées.

Transférez le programme dans la carte Arduino. La LED doit s’éclairer 1 fois toutes les 2s pendant 1s.

La période T du signal de commande de la LED se calcul à l’aide de : **T(s) = 0,5.10-6.a.b**

a et b sont les nombres placés dans les registres r27, r26, r25 et r24 tel que : **r27r26 ← a** et **r25r24 ← b**

**Q7.** Complétez le tableau ci-dessous.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **F(Hz)** | **T(s)** | **a.b** | **r27r26** | **r25r24** | **Remarque** |
| *0,5* | *2* | *4 .106* | *1000* | *4000* | *La LED s’éclaire 1 fois toute les 2s pendant 1s.* |
| *0x03E8* | *0x0FA0* |
| 1 | 1 |  |  |  | *La LED s’éclaire 1 fois par seconde pendant 0,5s.* |
|  |  |
| 2 | 0,5 |  |  |  | *La LED s’éclaire 2 fois par seconde pendant 0,25s.* |
|  |  |
| 4 | 0,25 |  |  |  | *La LED s’éclaire 4 fois par seconde pendant 0,125s.* |
|  |  |

* **Blink.asm dans le simulateur Wokwi**

Exemple de programmation en assembleur dans le simulateur Wokwi. À voir sur (<https://urlr.me/kc8FQ> )

**Q8.** Donnez le rôle des structures internes au CPU représenté ci-contre.

|  |  |
| --- | --- |
| * **Registre** : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ * **ALU** : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ * **Décodeur d’instructions** : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   * **Compteur programme (PC)** : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   * **Registre d’état (SREG)** : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |

**Résumé**

*« Le langage machine et le langage assembleur sont deux concepts fondamentaux en programmation informatique, particulièrement liés aux niveaux les plus bas de l’architecture des ordinateurs.*

***Langage Machine***

*Le langage machine est le langage natif d’un ordinateur, constitué de suites de bits (0 et 1) qui représentent des instructions exécutables directement par le processeur. Chaque type de processeur a son propre jeu d’instructions en langage machine, ce qui le rend difficile à lire et à écrire pour les humains.*

***Langage Assembleur***

*Le langage assembleur, ou langage d’assemblage, est une représentation plus lisible par l’homme du langage machine*[*2*](https://fr.wikipedia.org/wiki/Assembleur)*. Il utilise des mnémoniques, des symboles faciles à retenir, pour représenter les instructions du processeur. Par exemple, au lieu d’écrire une instruction en binaire, on utilise des mots comme LDI (pour charger des données) ou ADD (pour additionner des valeurs).*

***Relation entre langage machine et assembleur***

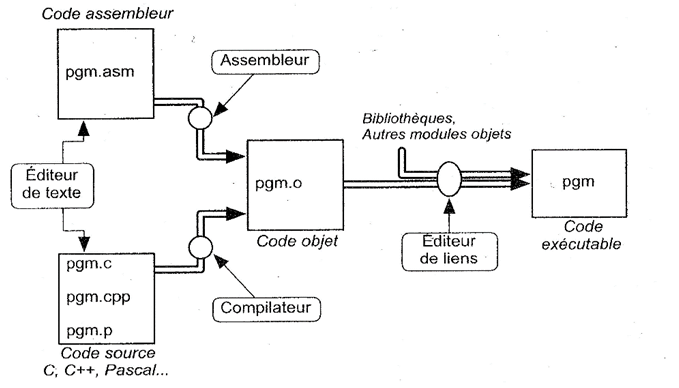
*Le langage assembleur doit être converti en langage machine par un programme appelé assembleur. Cette conversion permet de créer des fichiers exécutables ou des fichiers objets qui peuvent être directement exécutés par l’ordinateur.*

***Utilisation***

*Historiquement, le langage assembleur était largement utilisé pour écrire des programmes nécessitant une grande efficacité et un contrôle précis du matériel, comme les systèmes d’exploitation et les logiciels embarqués. Aujourd’hui, il est souvent supplanté par des langages de programmation de plus haut niveau, mais reste crucial pour certaines applications spécifiques telles que les pilotes de périphériques. »*

*Source :* [*Développez.com*](https://benoit-m.developpez.com/assembleur/tutoriel/)

***Cycle d’un programme compilé ou assemblé***

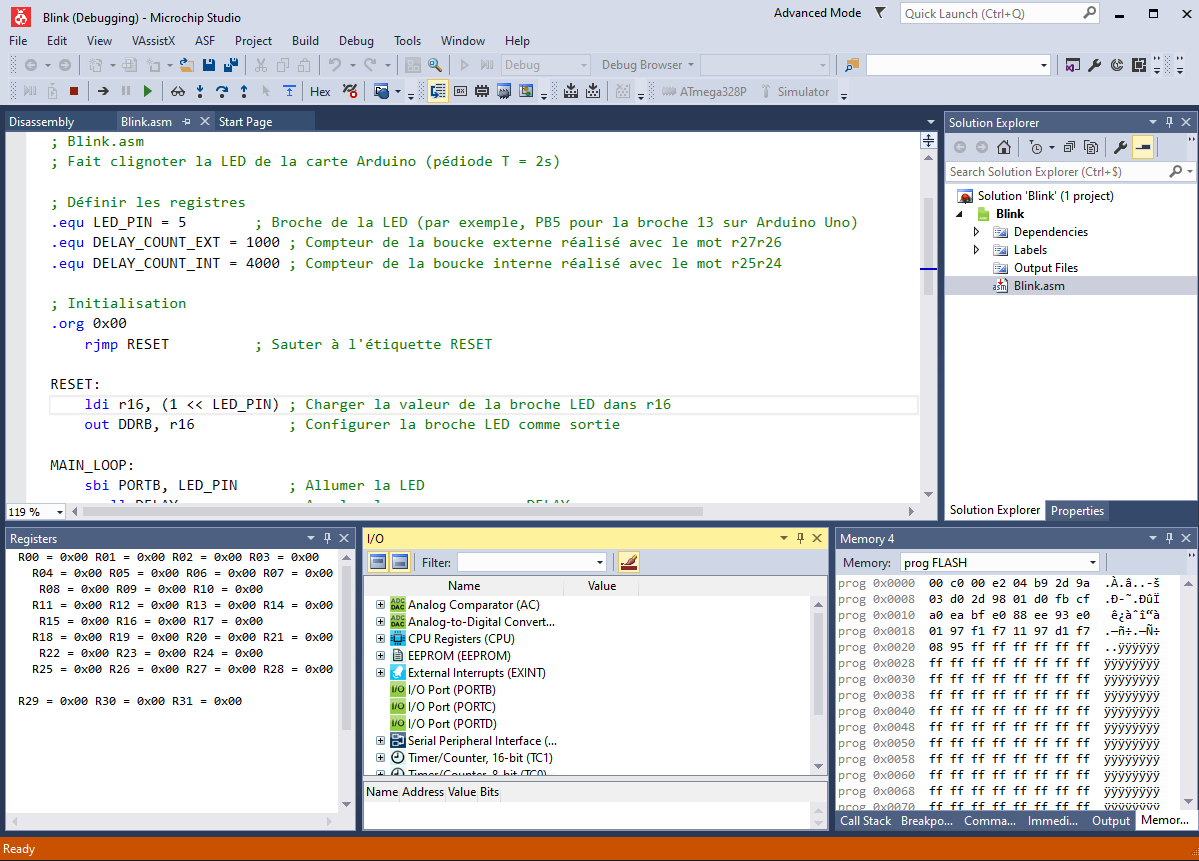
****

**ANNEXES**

* **Architecture matérielle de** [**l’ATMEGA328P**](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf)

|  |  |
| --- | --- |
| **Architecture AVR** | **Mémoire programme (flash)** |
|  |  |
| **Mémoire de données (SRAM)** |
|  |

* **L’IDE Microchip Studio**



**Pour aller plus loin**

Microcontrôleur Atmega 328P et sa programmation en langage assembleur (Module 1) https://urlr.me/tzdng

Wokwi - Jeu de pong en assembleur AVR https://urlr.me/kfXVY