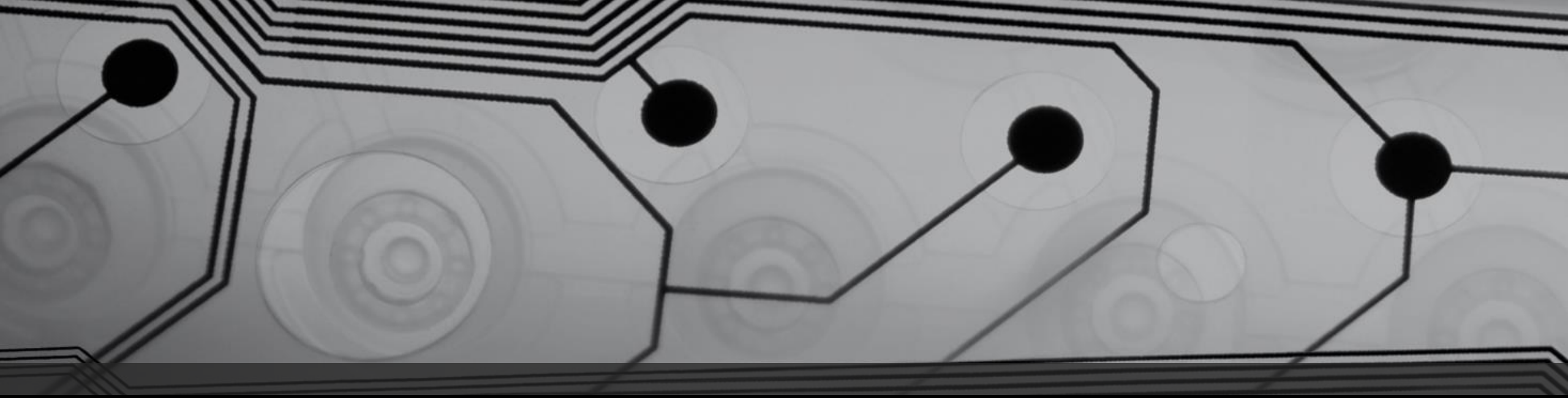


Langage machine et assembleur

Découverte

Sommaire

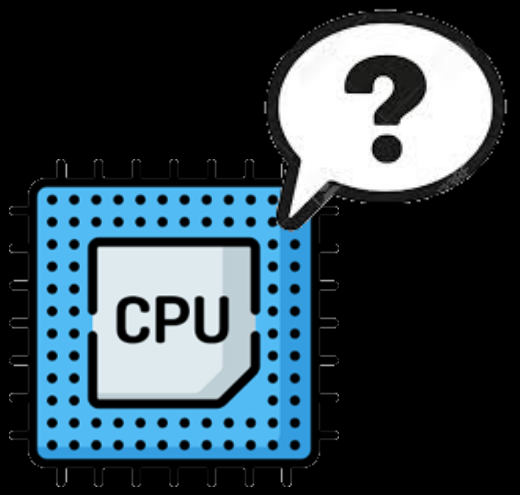
J'écris le **code source** de mon programme avec un langage de **haut** ou de **bas niveau** !

Indice TIOBE en Août 2024

Programming Language	Ratings	Change
Python	18.04%	+4.71%
C++	10.04%	-0.59%
C	9.17%	-2.24%
Java	9.16%	-1.16%
C#	6.39%	-0.65%
JavaScript	3.91%	+0.62%
SQL	2.21%	+0.68%
Visual Basic	2.16%	-0.45%
Go	2.03%	+0.87%
Assembly language	1.21%	-0.13%

Changements par rapport à août 2023.

On trouve l'assembleur à la 17e place.



1. Langages de haut niveau, de bas niveau et …

Le CPU "comprend"le langage machine !

J'exécute du **code machine** ou **"langage machine"** !

... langage machine ?

Le passage du **code source** au **code machine** nécessite une traduction :

Cas 1

Le code source est traduit en une seule opération à l'aide d'un **COMPILATEUR.**

Exemple : le code source C++ est compilé.

Cas 2

Le code source est traduit ligne par ligne à l'aide d'un **INTERPRETEUR.**

Exemple : le code source JavaScript est interprété.

Exemple : le « Hello World ! » pour l'embarqué

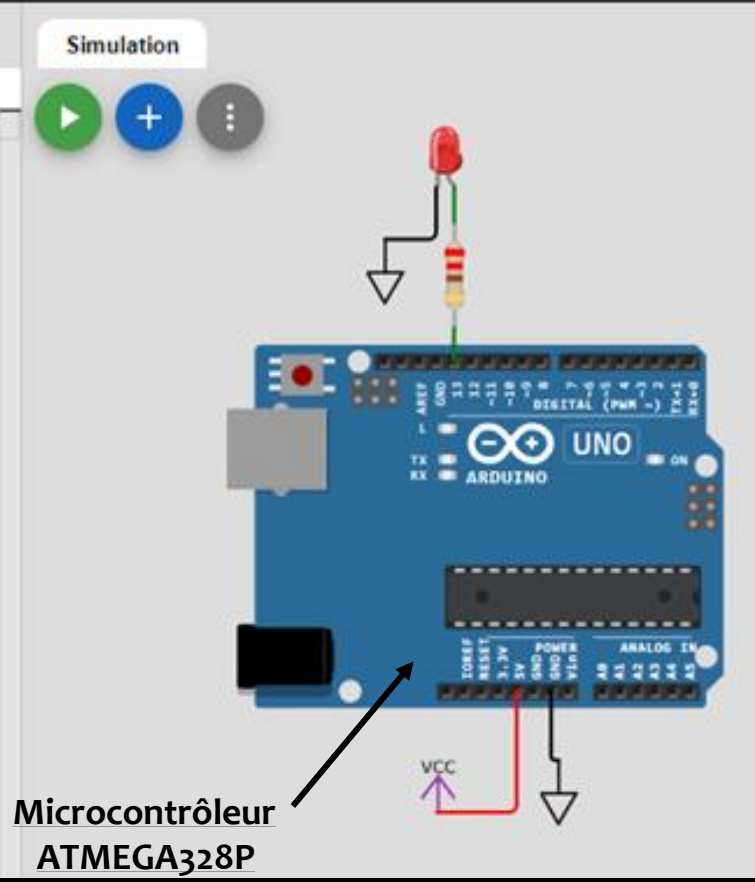
WOKWI SAVE SHARE Blink

sketch.ino | diagram.json | Library Manager

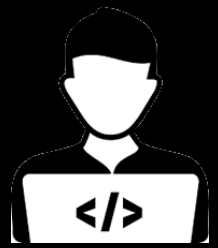
```ino
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Code source écrit en **C++** (Arduino)

« Hello World ! » embarqué <=> faire clignoter une led !



Microcontrôleur ATMEGA328P



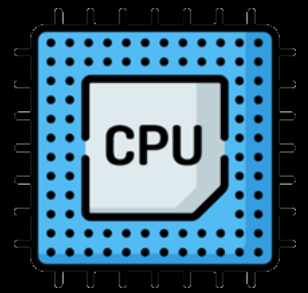
2. Comment traduire du code source de haut niveau en code machine ? – 2.2 "Hello World !" [Wokwi]

Code machine de « Hello World ! »

Le **compilateur** intégré à l'outil de développement (IDE) **MICROCHIP STUDIO** produit un fichier ***Blink.hex*** à partir du code source contenu dans ***Blink.cpp***.

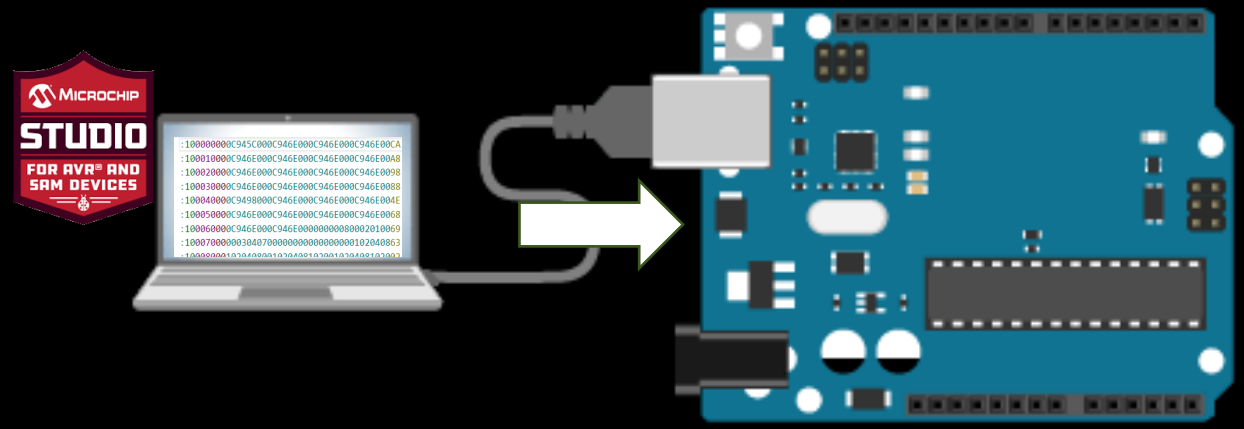

```
:100000000C945C000C946E000C946E000C946E00CA
:100010000C946E000C946E000C946E000C946E00A8
:100020000C946E000C946E000C946E000C946E0098
:100030000C946E000C946E000C946E000C946E0088
:100040000C949B000C946E000C946E000C946E004E
:100050000C946E000C946E000C946E000C946E0068
:100060000C946E000C946E0000000000080002010069
:10007000000304070000000000000000102040863
:1000800010204080010204081020010204081020
```

Blink.hex contient le **code machine** et des informations utiles à la **programmation** du microcontrôleur.



Transfert du code machine de « Hello World ! » dans la mémoire du microcontrôleur

MICROCHIP STUDIO transfère seulement le code machine dans le microcontrôleur.

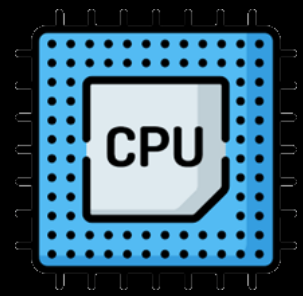


Exemple : code machine (**partiel**) du programme Blink en mémoire

Memory 4

Memory: prog FLASH

```
prog 0x0000  0c 94 5c 00 0c 94 6e 00 0c 94 6e 00 0c 94 6e  .”\..”n..”n..”n
prog 0x000F  00 0c 94 6e 00 0c 94 6e 00 0c 94 6e 00 0c 94  ..”n..”n..”n..”
prog 0x001E  6e 00 0c 94 6e 00 0c 94 6e 00 0c 94 6e 00 0c  n..”n..”n..”n..
```



Le langage de bas niveau – L'assembleur

*« Le langage d'assemblage ou **assembleur** est le langage de plus **bas niveau** représentant le **langage machine** sous une forme lisible par un humain. En assembleur, le **code source** est constitué de symboles dits "mnémoniques", c'est-à-dire faciles à retenir.*

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 **DESASSEMBLEUR.**

Exemple : extrait de Blink désassemblé

```
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
...
```

3. L'assembleur : langage de bas niveau ! 3.1 Désassembler le code machine de « Hello World ! »

Les mnémoniques sont décrits dans une documentation mais leur compréhension nécessite de connaître l'architecture matérielle du CPU.

Exemple extrait de AVR Instruction Set Manual

LDI – Load Immediate

Description

Loads an 8-bit constant directly to register 16 to 31.

Operation:

(i) Rd ← K

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

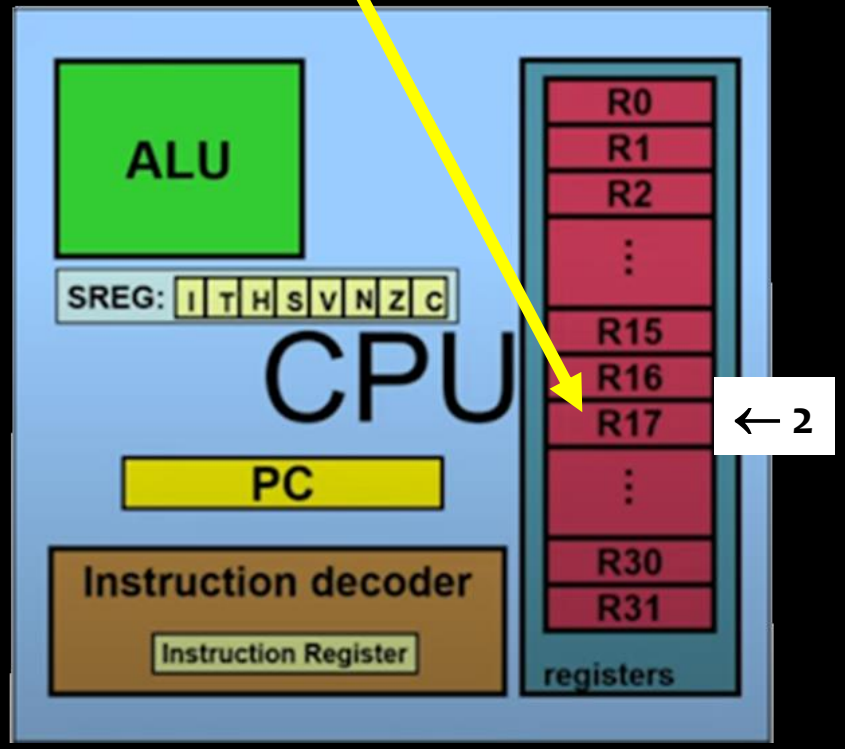
Example:

```
clr r31 ; Clear Z high byte
ldi r30,$F0 ; Set Z low byte to $F0
lpm ; Load constant from Program
; memory pointed to by Z
```

Words	1 (2 bytes)
Cycles	1

Code machine

Exemple : 12 e6 LDI R17,0x62 ; R17 ← 2



CPU de l'ATMEGA328P

3. L'assembleur : langage de bas niveau !

Les instructions "machine"

Elles peuvent être rangées en quatre catégories :

- les instructions **arithmétiques et logiques**
  Exemple : ADC, SBC, AND, OR, etc.
- les instructions de **transfert de données**
  Exemple : LD, LDI, MOV, etc.
- les instructions **d'entrées-sorties**
  Exemple : IN, OUT, etc.
- les instructions de **rupture de séquence**
  Exemple : BREQ, JMP, etc.

Remarques : les exemples font référence à la famille des processeurs AVR ATMEL (8bits).
L'ATMEGA328P de la carte ARDUINO UNO possède un jeu de 131 instructions.

« Hello World ! » réécrit en assembleur

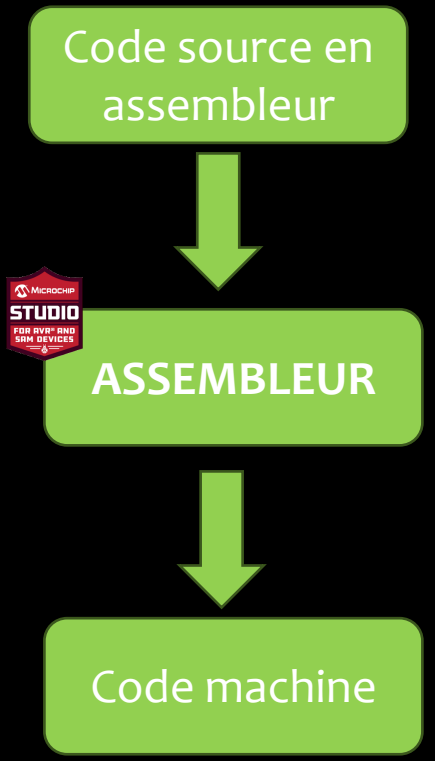
Outil logiciel - Assembleur

Un programme appelé **ASSEMBLEUR** convertit le code source écrit en assembleur en **code machine**.
L'IDE **Microchip Studio** dispose d'un assembleur**.**

Le code source assembleur est écrit par le programmeur.

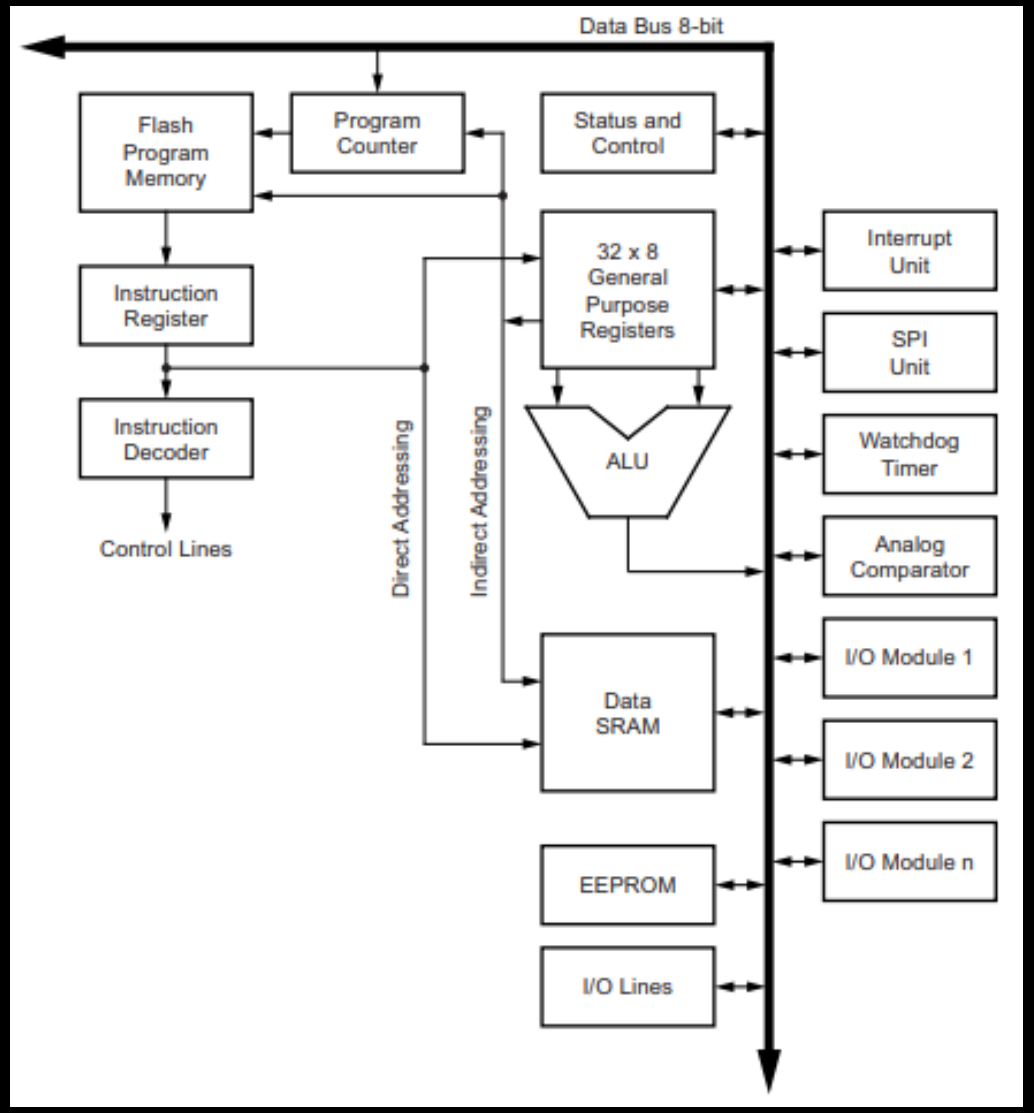
```
; Blink.asm
; Fait clignoter la LED de la carte Arduino (pédiode T = 2s)

; Définir les registres
.equ LED_PIN = 5        ; Broche de la LED (par exemple, PB5 pour la broche 13 sur Arduino Uno)
.equ DELAY_COUNT_EXT = 1000 ; Compteur de la boucke externe
.equ DELAY_COUNT_INT = 4000 ; Compteur de la boucke interne

; Initialisation
.org 0x00
    rjmp RESET          ; Sauter à l'étiquette RESET

RESET:
    ldi r16, (1 << LED_PIN) ; Charger la valeur de la broche LED dans r16
    out DDRB, r16           ; Configurer la broche LED co

MAIN_LOOP:
    sbi PORTB, LED_PIN      ; Allumer la LED
    rcall DELAY             ; Appeler le sous programme D
    cbi PORTB, LED_PIN      ; Éteindre la LED
    rcall DELAY             ; Appeler le sous programme DELAY
    rjmp MAIN_LOOP          ; Sauter à l'étiquette MAIN_LOOP (Boucle principale)
```

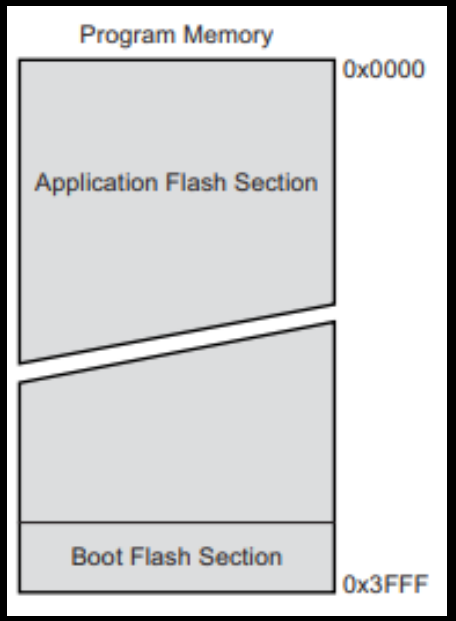
```
Memory:  prog FLASH
prog 0x0000   00 c0 00 e2 04 b9 2d 9a 03 d0 2d 98 01 d0 fb cf
prog 0x0010   80 ea 9f e0 a8 ee b3 e0 01 97 f1 f7 11 97 d1 f7
prog 0x0020   08 95 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

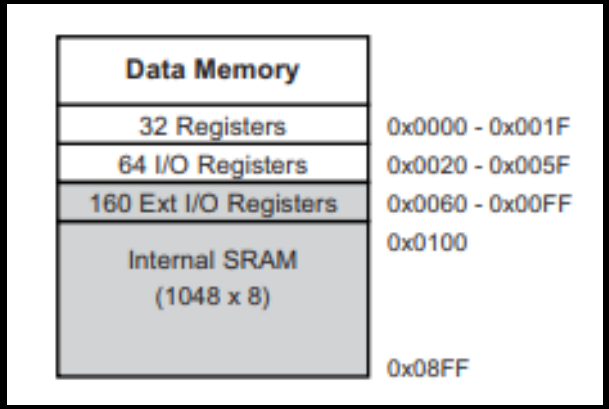


3. L'assembleur : langage de bas niveau ! 3.2 Réécrire « Hello World ! » en assembleur [Wokwi]

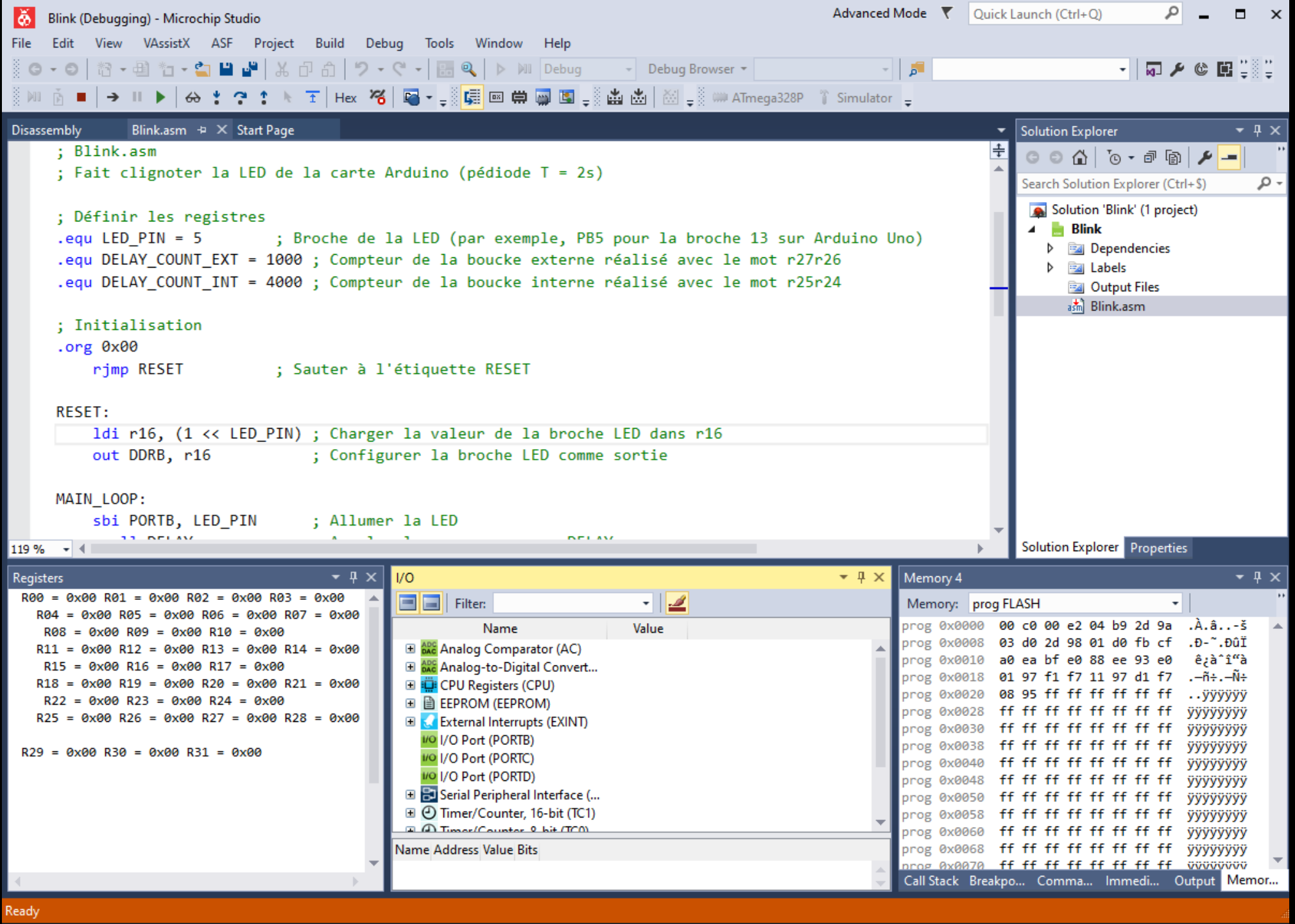
Annexe 1 – Architecture matérielle AVR







MICROCHIP STUDIO



ANNEXE 2

A3. Jeu de Pong en assembleur sur Arduino

```
52
53  ; ####### Data area ###### (stored in SRAM, starts above the register area)
54          .data
55
56  ;(mostly just to test load/store from SRAM instructions)
57  tmr1:   .word  0          ; A "timer"
58  bats:   .byte  0          ; bat positions (center [2-7]), one nibble for each
59                            ; bit 4 in top nibble is direction (as position never exceeds 7)
60
61  ; extended version will have the display buffer in SRAM to allow for more display units)
62
63  ; ###### Code area ########  (Storing code in flash)
64          .text
65
66  ; ==== send one byte-pair to display (partial transaction)
67  SendTwo:
68      ; entry: byte values in Tmp2/3 (r20/21)
69      ; exit: Tmp altered
70      out   SPDR,Tmp2       ; send the first byte, the row - put it in SPI data register
71  1: in    Tmp,SPSR         ; Read SPI status register
72      sbrs  Tmp,SPIF        ; was the done bit set ?
73      rjmp  1b              ; nope, keep waiting
74
75      out   SPDR,Tmp3       ; 2nd byte Display pattern - send it too
76  1: in    Tmp,SPSR         ; Read SPI status register
77      sbrs  Tmp,SPIF        ; was the done bit set ?
78      rjmp  1b              ; nope, keep waiting
79
80      ret
81
82  ; ===== send one row to display
83  SendRow:
84      ; Push four bytes to SPI, bracket by LOW pin 10
```