

TP3 : MICROPROCESSOR

EX.1 Structure de base

Structure d'un microprocesseur 4 bit :

Unité de traitement (ALU), unité de mémoire (Register Bank), accumulateur, unité de control, MUX4 :1.

Fonctionnement :

Le microprocesseur exécute une suite d'instructions contenues en mémoire. La mémoire contient d'une part les instructions (le programme), d'autre part des données (par exemple une image ou un son, ou un fichier texte, à traiter par un programme). L'unité de contrôle analyse les instructions une par une, et pour chacune, indique à l'unité de traitement ce qu'elle doit faire en activant les signaux de contrôle adéquats. Les instructions (par exemple une addition) sont exécutées par l'unité de traitement

Comment sont générés les signaux de contrôle pour exécuter des instructions :

Ils sont générés par l'unité de contrôle, elle envoie des commandes à l'unité de traitement et de mémoire qui vont exécuter les instructions.

Code :

```

21 // Declaration of the variables of the Micro-processor
22 bool Wr_ACC, I[4], op, Wr_RB, Adr[2], sel, Clk;
23
24 Reg Reg0, Reg1, Reg2, Reg3, ACC;
25 Output4 ALU4, RB, MUX;
26 ADD_SUB ADD4, SUB4;
27
28 // Assign output variables to GPIO pins
29 const int Led_O0 = 26;
30 const int Led_O1 = 27;
31 const int Led_O2 = 14;
32 const int Led_O3 = 12;
33
34 void setup()
35 {
36     // Initialise the variables
37     Wr_ACC = 0;
38     I[0]=0;
39     I[1]=0;
40     I[2]=0;
41     I[3]=0;
42     op = 0;
43     Wr_RB = 0;
44     Adr[0] = 0;
45     Adr[1] = 0;
46     sel = 0;
47     Clk = 0;
48
49     // Initialise the LED variables as outputs
50     pinMode(Led_O0, OUTPUT);
51     pinMode(Led_O1, OUTPUT);
52     pinMode(Led_O2, OUTPUT);
53     pinMode(Led_O3, OUTPUT);
54
55     // Set LED outputs to LOW
56     digitalWrite(Led_O0, LOW);
57     digitalWrite(Led_O1, LOW);
58     digitalWrite(Led_O2, LOW);
59     digitalWrite(Led_O3, LOW);
60
61     RB = Register_bank(ACC.O, Adr, Clk, Wr_RB);
62     ALU4 = ALU_4(ACC.O, RB.O, op);
63     MUX = MUX21(sel, I, ALU4.O);
64     ACC = Accumulator(MUX.O, Clk, Wr_ACC);
65
66     // LED outputs
67     digitalWrite(Led_O0, ACC.O[0]);
68     digitalWrite(Led_O1, ACC.O[1]);
69     digitalWrite(Led_O2, ACC.O[2]);
70     digitalWrite(Led_O3, ACC.O[3]);

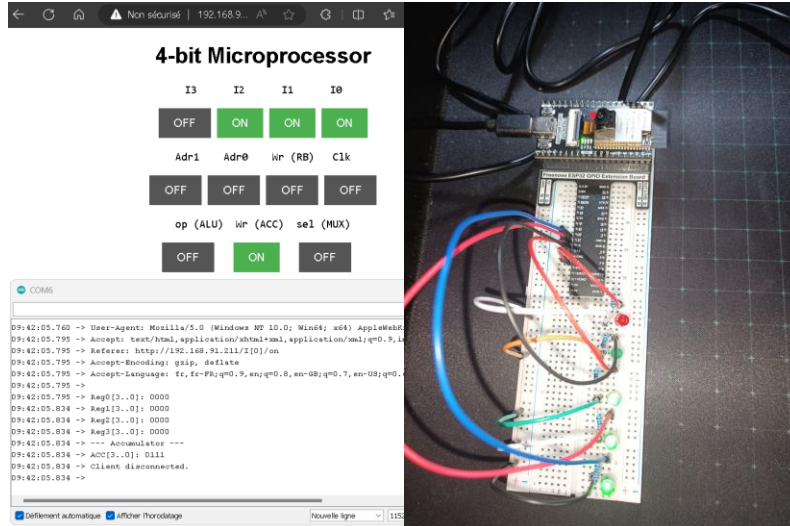
```

EX.1.4 Programmation :

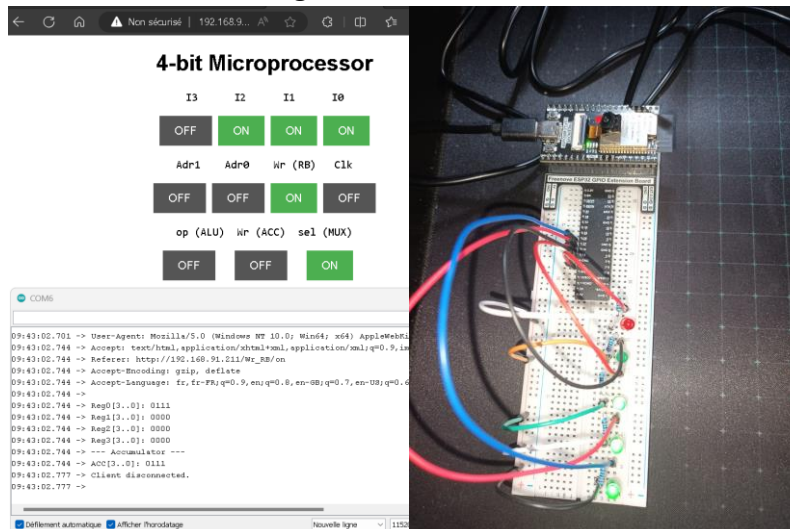
Expliquer également comment vous procédez cette opération :

Voici comment procéder pour l'opération 7+3-6 :

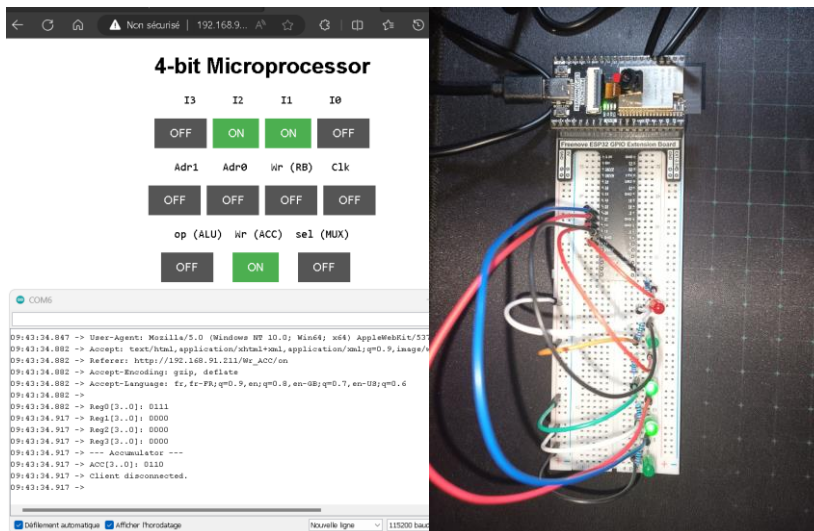
1. Charger 7 (0111) dans l'accumulateur :



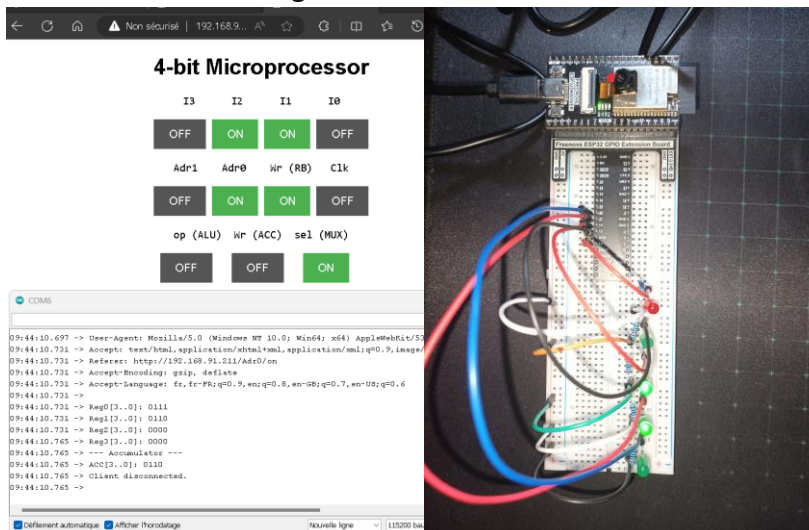
- 2.
3. Stocker le 7 dans le registre à l'adresse 00 :



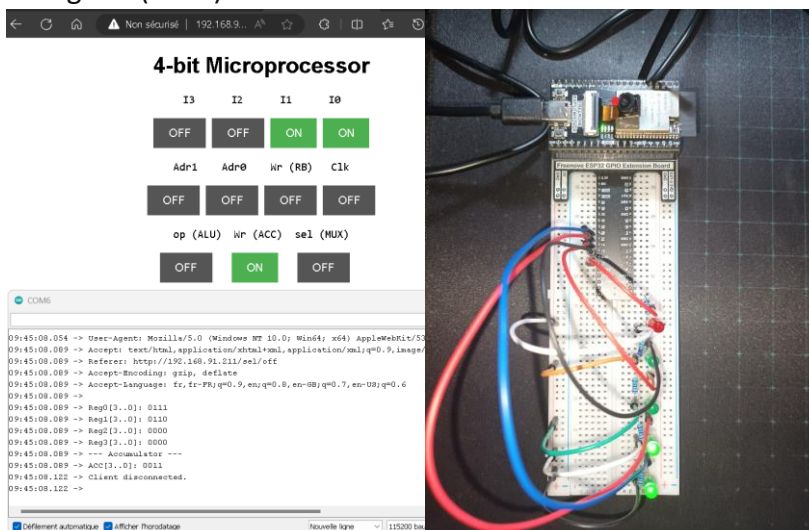
- 4.
5. Charger 6 (0110) dans l'accumulateur :



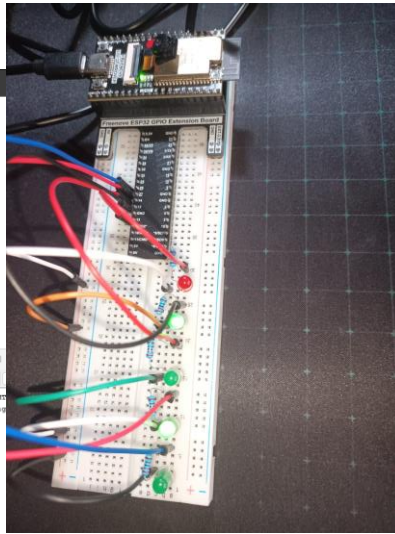
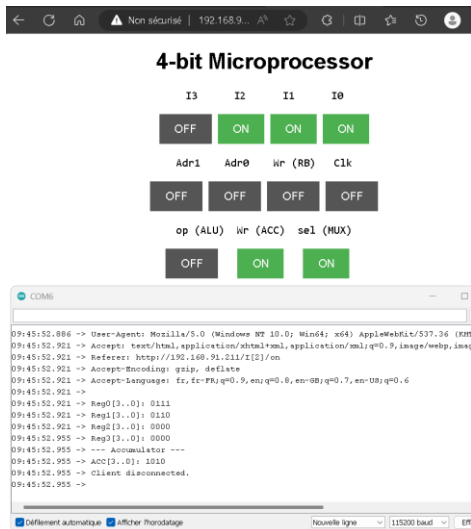
- 6.
7. Stocker le 6 dans le registre à l'adresse 01 :



- 8.
9. Charger 3 (0011) dans l'accumulateur :

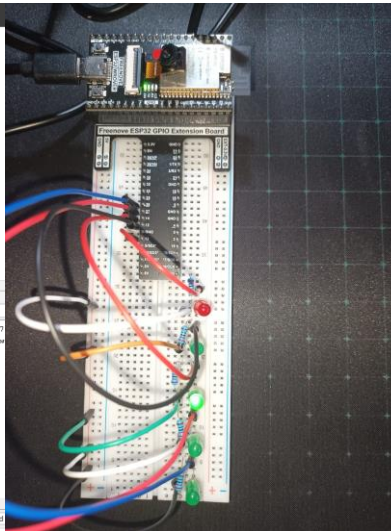
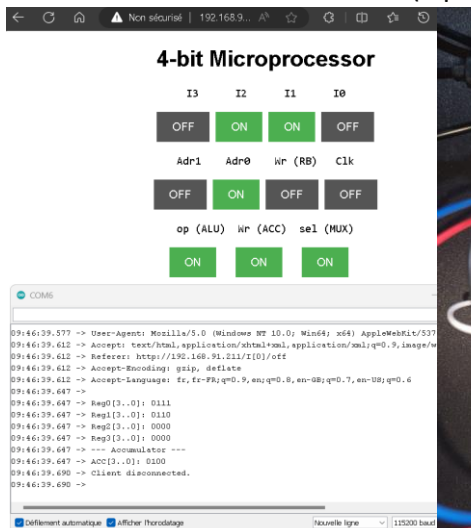


- 10.
11. Ajouter 7 à l'accumulateur (à partir de l'adresse 00) :



12.

13. Soustraire 6 de l'accumulateur (à partir de l'adresse 01) :



14.

TP4 : INSTRUCTIONS

Ex.1.1 Signaux de contrôle :

Comment sont générés les signaux de contrôle à partir de ces instructions binaires :

$Wr_acc = !b_5 + b_5 * b_4$

$Wr_acc = !Wr_BR$

$Sel = !b_5$

$Adr = b_3 * b_2$

$Op = b_4$

$E0 = b_3 * b_2 * b_1 * b_0$

Code :

```
1 #include <Wifi.h>
2 #include <AsyncTCP.h>
3 #include <ESPAsyncWebServer.h>
4 #include "circuits.h"
5
12 const char* PARAM_INPUT = "Instruction";
13 char instruction[7];
14
15
16 // Declaration of the variables of the Micro-processor
17 bool Wr_ACC, I[4], op, Wr_RB, Adr[2], sel, Clk;
18
19 Reg Reg0, Reg1, Reg2, Reg3, ACC;
20 Output4 ALU4, RB, MUX;
21 ADD_SUB ADD4, SUB4;
22
23 // Declaration of the variables for the instructions
24 bool b[6];
25
26 // Assign output variables to GPIO pins
27 const int Led_O0 = 26;
28 const int Led_O1 = 27;
29 const int Led_O2 = 14;
30 const int Led_O3 = 12;
```

```

49 void setup()
50 {
51     // Initialise the register variables
52
53     // Initialise the variables
54     Wr_ACC = 0;
55     I[0]=0;
56     I[1]=0;
57     I[2]=0;
58     I[3]=0;
59     op = 0;
60     Wr_RB = 0;
61     Adr[0] = 0;
62     Adr[1] = 0;
63     sel = 0;
64     Clk = 0;
65     b[0] = 0;
66     b[1] = 0;
67     b[2] = 0;
68     b[3] = 0;
69     b[4] = 0;
70     b[5] = 0;
180     b[0] = instruction[5] - 48;
181     b[1] = instruction[4] - 48;
182     b[2] = instruction[3] - 48;
183     b[3] = instruction[2] - 48;
184     b[4] = instruction[1] - 48;
185     b[5] = instruction[0] - 48;
186
189     I[0] = b[0];
190     I[1] = b[1];
191     I[2] = b[2];
192     I[3] = b[3];
193
194     // Compléter les signaux de controle
195     Wr_ACC = Ib[5] | b[5] & b[4];
196     Wr_RB = !Wr_ACC;
197     sel = Ib[5];
198     Adr[0] = I[2];
199     Adr[1] = I[3];
200     op = b[4];
201
202
203     //////////////////////////////////////
204
205     Clk = 1;
206
207     // Compléter le microprocesseur
208     RB = Register_bank(ACC.O, Adr, Clk, Wr_RB);
209     ALU4 = ALU_4(ACC.O, RB.O, op);
210     MUX = MUX21(sel, I, ALU4.O);
211     ACC = Accumulator(MUX.O, Clk, Wr_ACC);
212
213     Clk = 0;
214
215     // Compléter les indicateurs LEDs
216     digitalWrite(Led_O0, ACC.O[0]);
217     digitalWrite(Led_O1, ACC.O[1]);
218     digitalWrite(Led_O2, ACC.O[2]);
219     digitalWrite(Led_O3, ACC.O[3]);
220
221

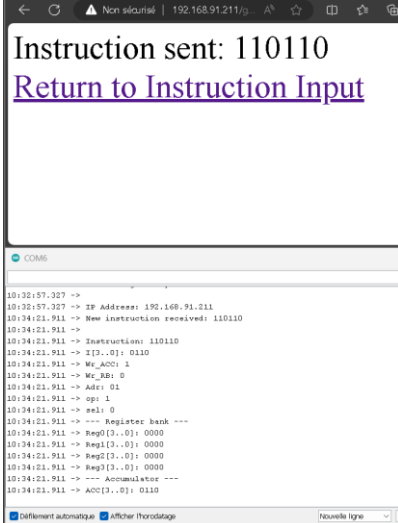
```

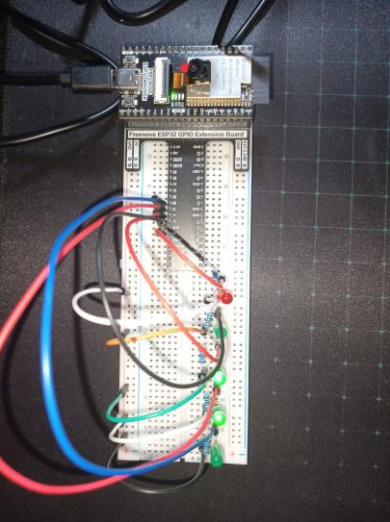
Ex.1.3 Programmation :

Instruction binaire 7+3-6 :

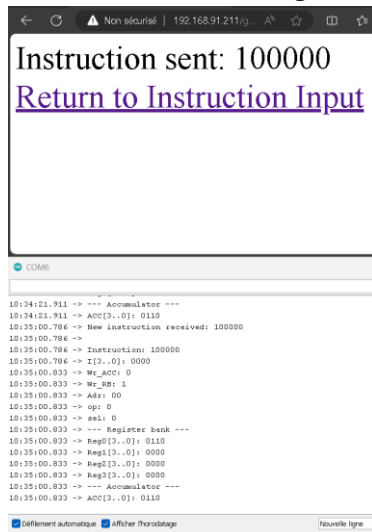
Voici comment procéder pour l'opération 7+3-6 :

1. Charger 6 (0110) dans l'accumulateur :



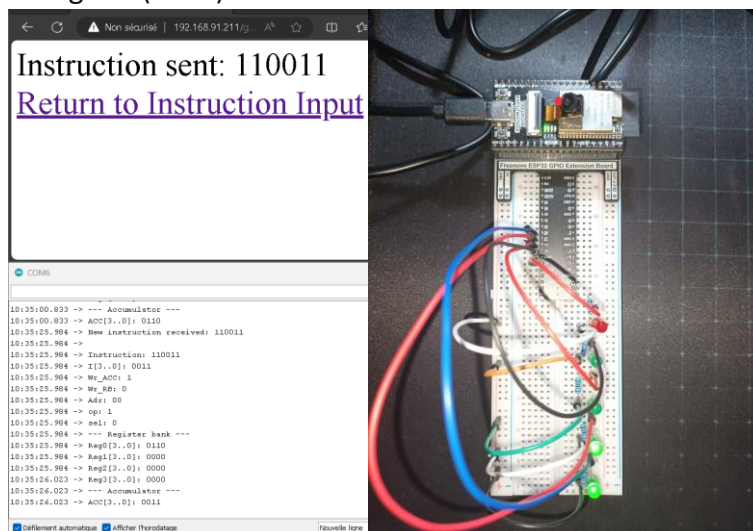


3. Stocker le 6 dans le registre à l'adresse 00 :



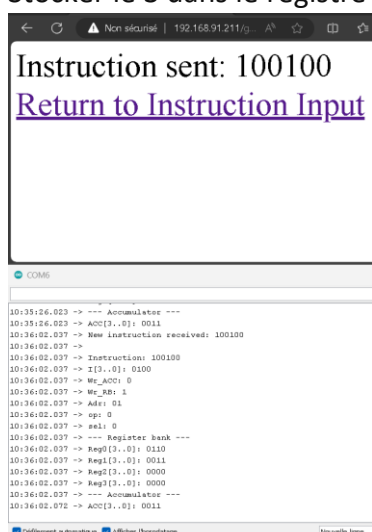
```
10:34:21.911 -> --- Accumulator ---
10:34:21.911 -> ACC[3..0]: 0110
10:35:00.786 -> New instruction received: 100000
10:35:00.786 -> Instruction: 100000
10:35:00.786 -> I[3..0]: 0000
10:35:00.833 -> Wt_ACC: 0
10:35:00.833 -> Wt_Rt: 1
10:35:00.833 -> Adt: 00
10:35:00.833 -> Opt: 0
10:35:00.833 -> sel: 0
10:35:00.833 -> --- Register bank ---
10:35:00.833 -> Reg0[3..0]: 0110
10:35:00.833 -> Reg1[3..0]: 0000
10:35:00.833 -> Reg2[3..0]: 0000
10:35:00.833 -> Reg3[3..0]: 0000
10:35:00.833 -> --- Accumulator ---
10:35:00.833 -> ACC[3..0]: 0110
```

4. ☐ Défilement automatique ☒ Afficher l'horodatage
5. Charger 3 (0011) dans l'accumulateur :



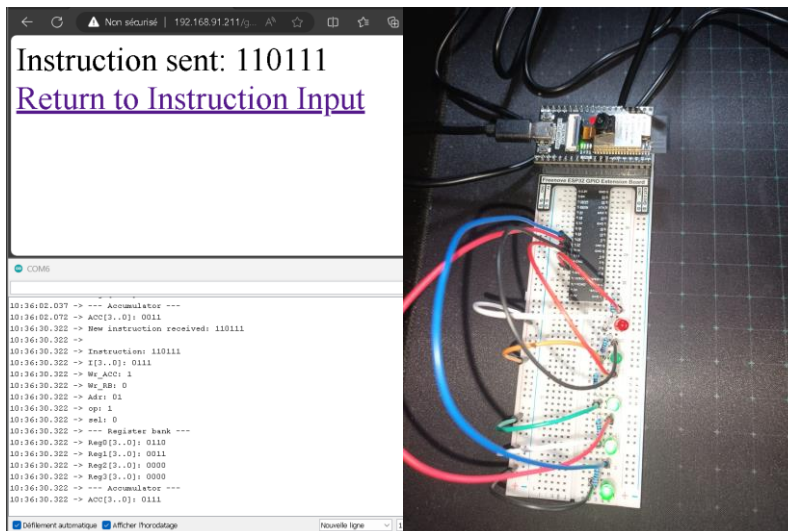
```
10:35:00.833 -> --- Accumulator ---
10:35:00.833 -> ACC[3..0]: 0110
10:35:25.984 -> New instruction received: 110011
10:35:25.984 -> Instruction: 110011
10:35:25.984 -> I[3..0]: 0011
10:35:25.984 -> Wt_ACC: 1
10:35:25.984 -> Wt_Rt: 0
10:35:25.984 -> Adt: 00
10:35:25.984 -> Opt: 1
10:35:25.984 -> sel: 0
10:35:25.984 -> --- Register bank ---
10:35:25.984 -> Reg0[3..0]: 0110
10:35:25.984 -> Reg1[3..0]: 0000
10:35:25.984 -> Reg2[3..0]: 0000
10:35:26.023 -> Reg3[3..0]: 0000
10:35:26.023 -> --- Accumulator ---
10:35:26.023 -> ACC[3..0]: 0011
```

6. ☐ Défilement automatique ☒ Afficher l'horodatage
7. Stocker le 3 dans le registre à l'adresse 01 :



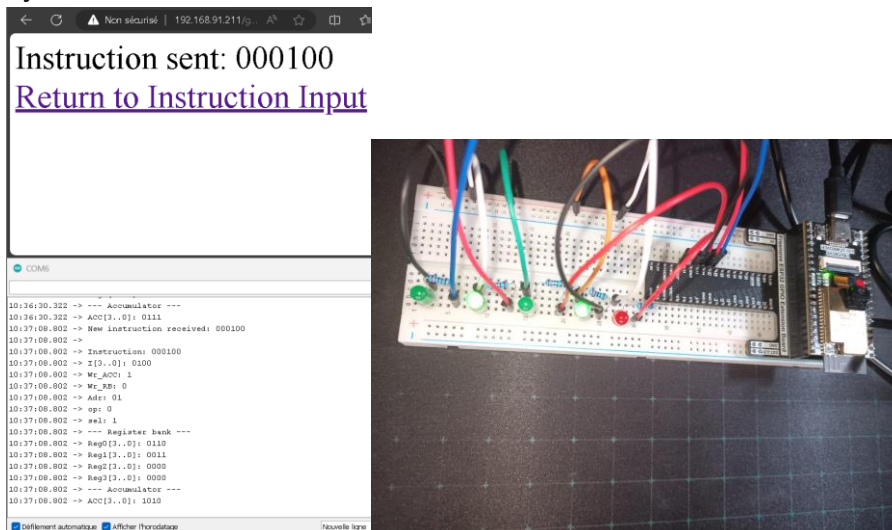
```
10:35:26.023 -> --- Accumulator ---
10:35:26.023 -> ACC[3..0]: 0011
10:36:02.037 -> New instruction received: 100100
10:36:02.037 -> Instruction: 100100
10:36:02.037 -> I[3..0]: 0100
10:36:02.037 -> Wt_ACC: 0
10:36:02.037 -> Wt_Rt: 1
10:36:02.037 -> Adt: 01
10:36:02.037 -> Opt: 0
10:36:02.037 -> sel: 0
10:36:02.037 -> --- Register bank ---
10:36:02.037 -> Reg0[3..0]: 0110
10:36:02.037 -> Reg1[3..0]: 0011
10:36:02.037 -> Reg2[3..0]: 0000
10:36:02.037 -> Reg3[3..0]: 0000
10:36:02.037 -> --- Accumulator ---
10:36:02.037 -> ACC[3..0]: 0011
```

8. ☐ Défilement automatique ☒ Afficher l'horodatage
9. Charger 7 (0111) dans l'accumulateur :



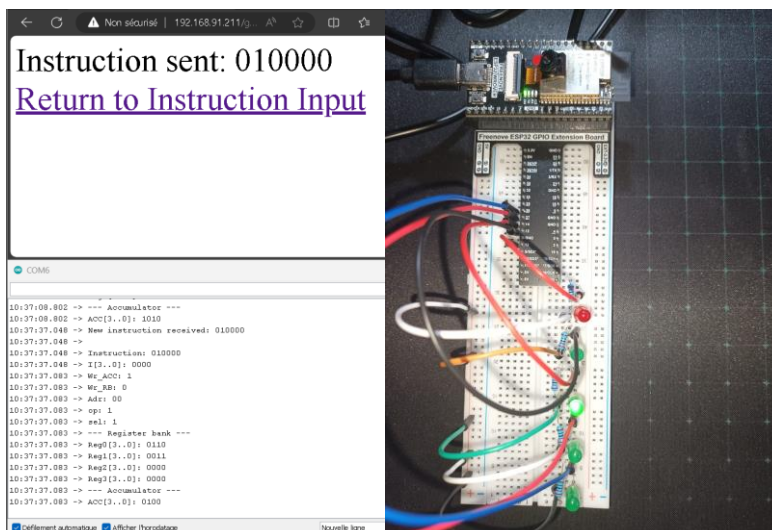
10.

11. Ajouter 3 à l'accumulateur :



12.

13. Soustraire 6 à l'accumulateur :



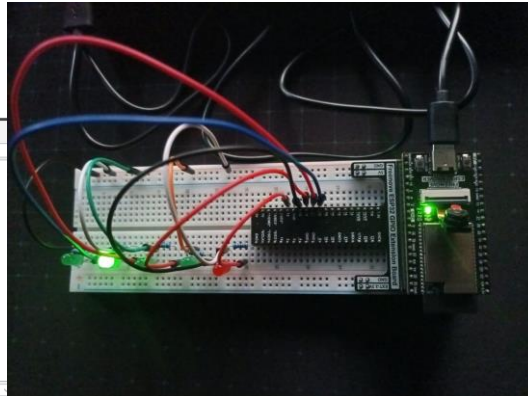
14.

Instruction binaire 7+3-6+5-2 :

Voici comment procéder pour l'opération 7+3-6+5-2 :

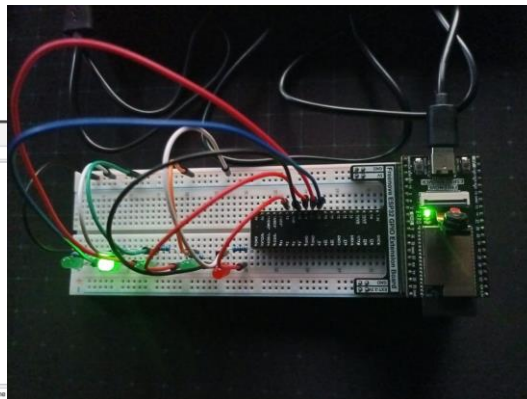
15. Charger 2 (0010) dans l'accumulateur :


```
Non sécurisé | 192.168.91.211/g...  
Instruction sent: 110010  
Return to Instruction Input  
COM6  
10:39:35.616 ->  
10:39:35.616 -> IP Address: 192.168.91.211  
10:39:57.340 -> New instruction received: 110010  
10:39:57.340 ->  
10:39:57.340 -> Instruction: 110010  
10:39:57.340 -> I[3..0]: 0010  
10:39:57.375 -> Wz_ACC: 1  
10:39:57.375 -> Wz_RB: 0  
10:39:57.375 -> AdR: 00  
10:39:57.375 -> op: 1  
10:39:57.375 -> sel: 0  
10:39:57.375 -> --- Register bank ---  
10:39:57.375 -> Reg0[3..0]: 0000  
10:39:57.375 -> Reg1[3..0]: 0000  
10:39:57.375 -> Reg2[3..0]: 0000  
10:39:57.375 -> Reg3[3..0]: 0000  
10:39:57.375 -> --- Accumulator ---  
10:39:57.375 -> ACC[3..0]: 0010  
Défilement automatique Afficher l'horodatage Nouvelle ligne
```



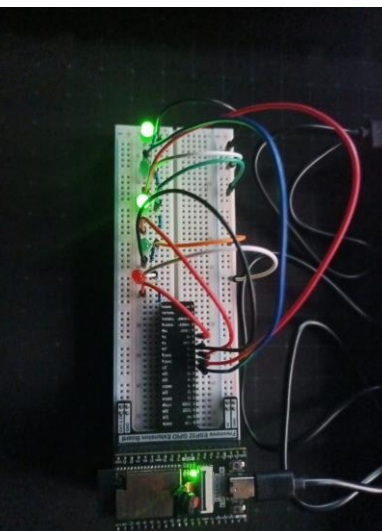
16.
17. Stocker le 2 dans le registre à l'adresse 00 :

```
Non sécurisé | 192.168.91.211/g...  
Instruction sent: 100000  
Return to Instruction Input  
COM6  
10:39:57.375 -> --- Accumulator ---  
10:39:57.375 -> ACC[3..0]: 0010  
10:40:43.845 -> New instruction received: 100000  
10:40:43.845 ->  
10:40:43.845 -> Instruction: 100000  
10:40:43.845 -> I[3..0]: 0000  
10:40:43.845 -> Wz_ACC: 0  
10:40:43.845 -> Wz_RB: 1  
10:40:43.845 -> AdR: 00  
10:40:43.845 -> op: 0  
10:40:43.845 -> sel: 0  
10:40:43.845 -> --- Register bank ---  
10:40:43.845 -> Reg0[3..0]: 0010  
10:40:43.845 -> Reg1[3..0]: 0000  
10:40:43.845 -> Reg2[3..0]: 0000  
10:40:43.845 -> Reg3[3..0]: 0000  
10:40:43.845 -> --- Accumulator ---  
10:40:43.845 -> ACC[3..0]: 0010  
Défilement automatique Afficher l'horodatage Nouvelle ligne
```

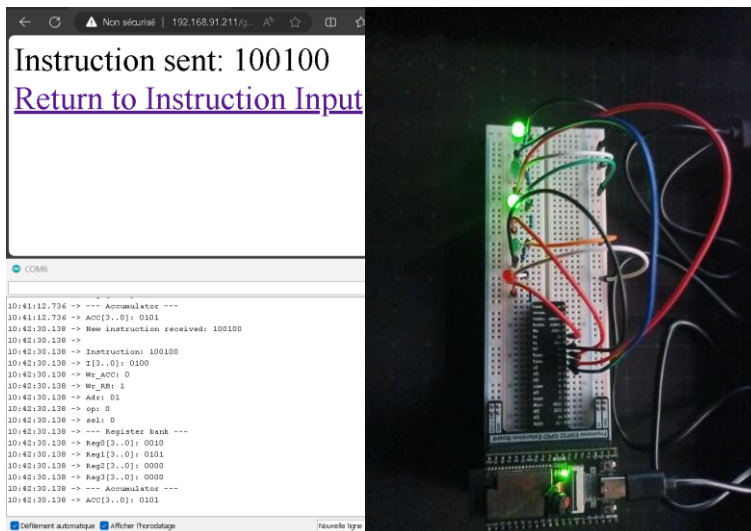


18.
19. Charger 5 (0101) dans l'accumulateur :

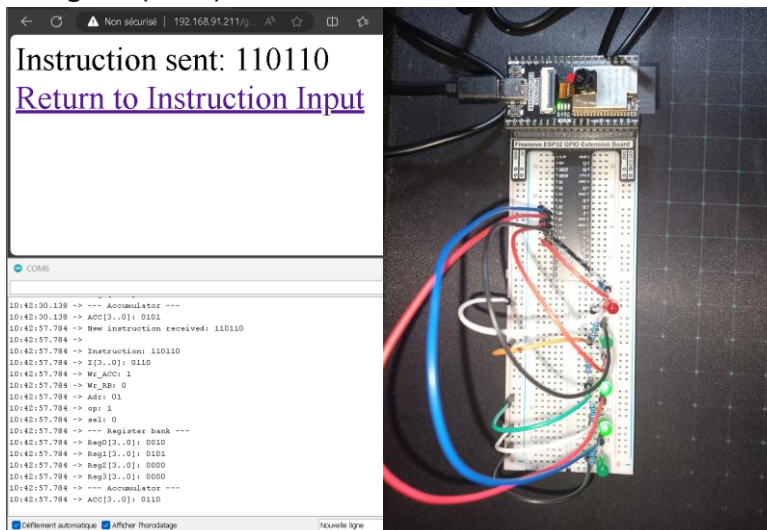
```
Non sécurisé | 192.168.91.211/g...  
Instruction sent: 110101  
Return to Instruction Input  
COM6  
10:40:43.845 -> --- Accumulator ---  
10:40:43.845 -> ACC[3..0]: 0010  
10:41:12.736 -> New instruction received: 110101  
10:41:12.736 ->  
10:41:12.736 -> Instruction: 110101  
10:41:12.736 -> I[3..0]: 0101  
10:41:12.736 -> Wz_ACC: 1  
10:41:12.736 -> Wz_RB: 0  
10:41:12.736 -> AdR: 01  
10:41:12.736 -> op: 1  
10:41:12.736 -> sel: 0  
10:41:12.736 -> --- Register bank ---  
10:41:12.736 -> Reg0[3..0]: 0010  
10:41:12.736 -> Reg1[3..0]: 0000  
10:41:12.736 -> Reg2[3..0]: 0000  
10:41:12.736 -> Reg3[3..0]: 0000  
10:41:12.736 -> --- Accumulator ---  
10:41:12.736 -> ACC[3..0]: 0101  
Défilement automatique Afficher l'horodatage Nouvelle ligne
```



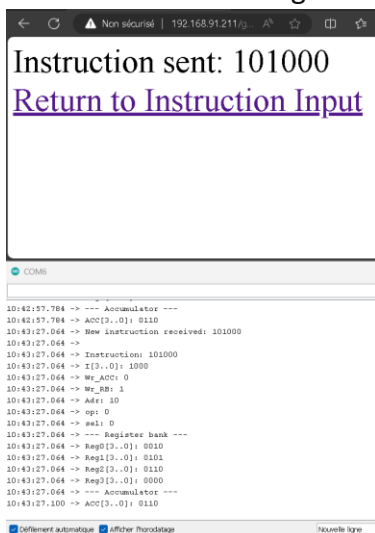
20.
21. Stocker le 5 dans le registre à l'adresse 01 :



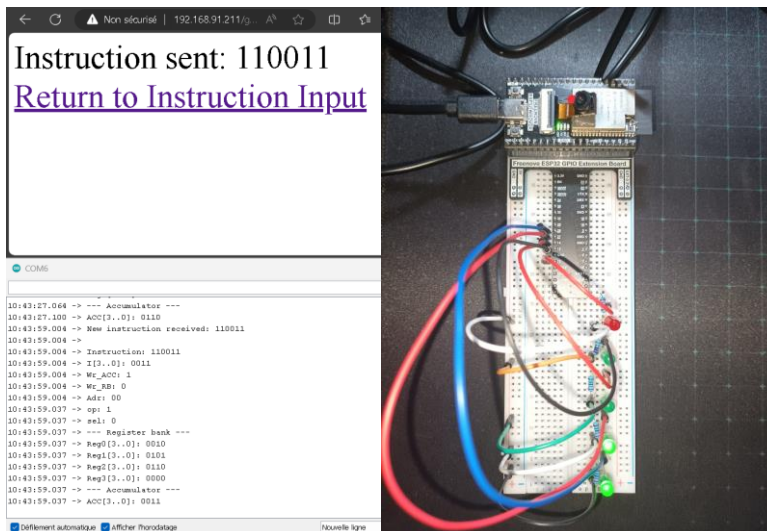
- 22.
23. Charger 6 (0110) dans l'accumulateur :



- 24.
25. Stocker le 6 dans le registre à l'adresse 10 :

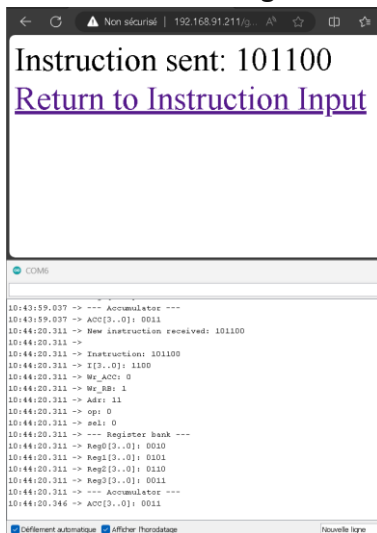


- 26.
27. Charger 3 (0110) dans l'accumulateur :



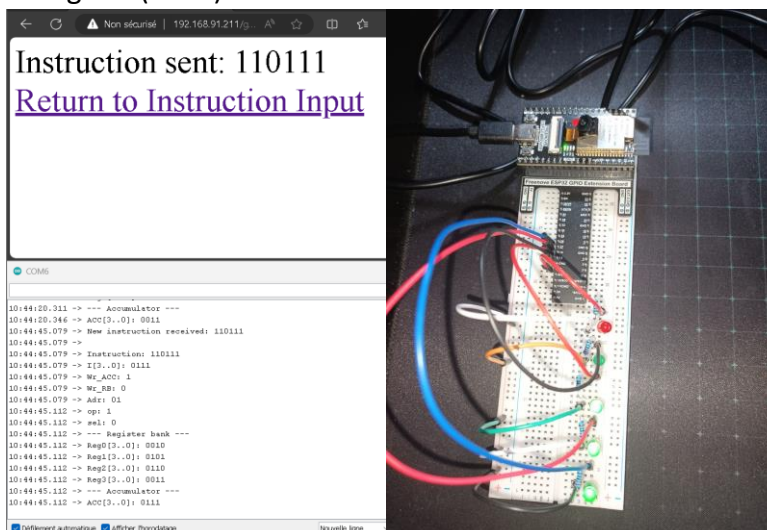
28.

29. Stocker 3 dans le registre à l'adresse 11 :



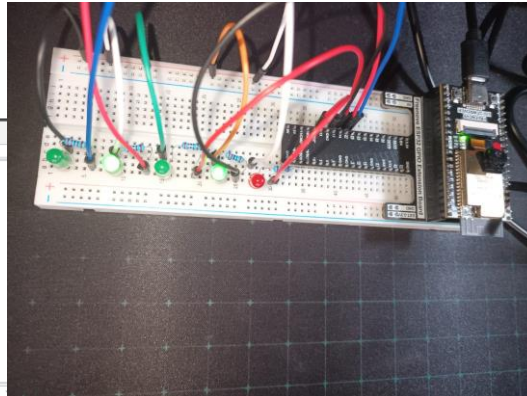
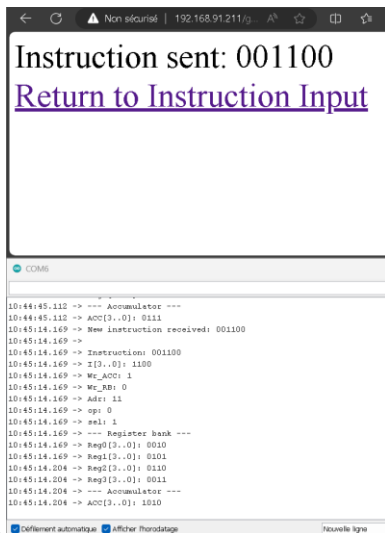
30.

31. Charger 7 (0111) dans l'accumulateur :

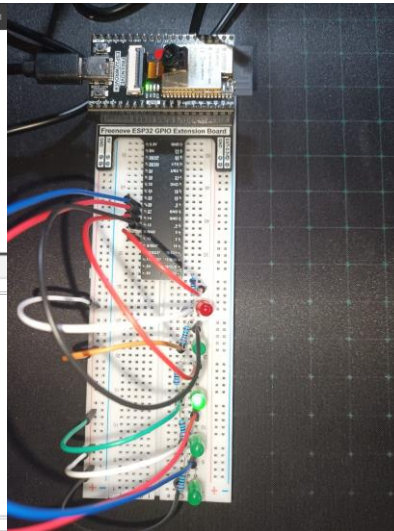
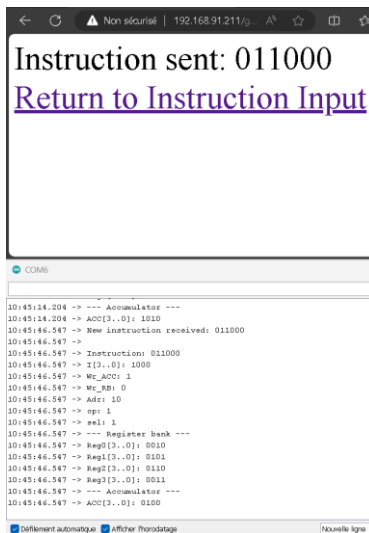


32.

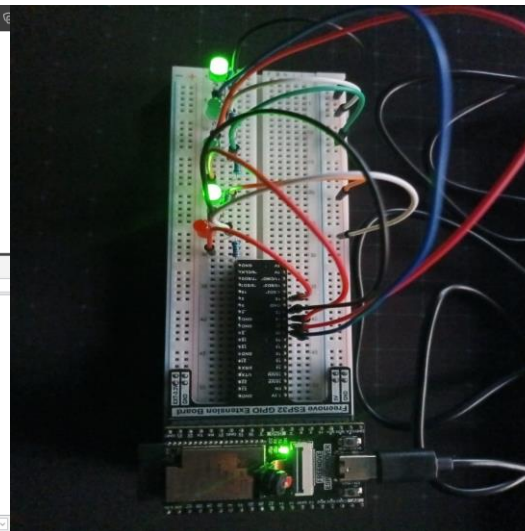
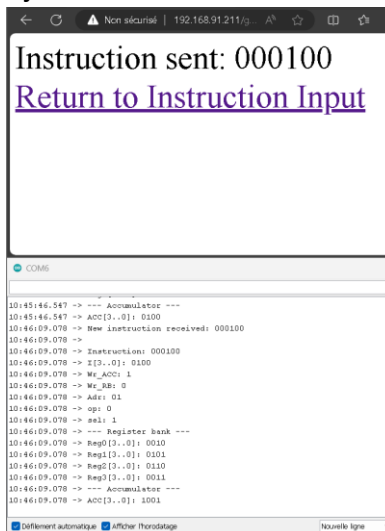
33. Ajouter 3 à l'accumulateur :



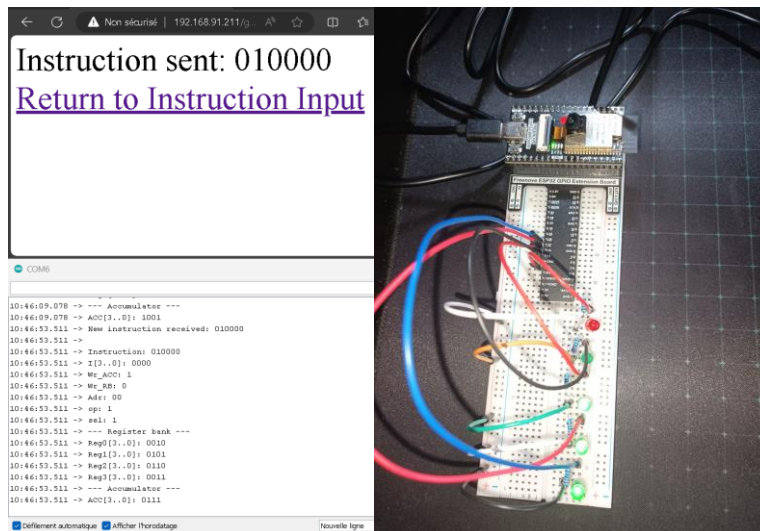
- 34.
35. Soustraire 6 à l'accumulateur :



- 36.
37. Ajouter 5 à l'accumulateur :



- 38.
39. Soustraire 2 à l'accumulateur :



40.

Le résultat final (7) devrait maintenant être dans l'accumulateur.