

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_8UB ADD4, SUB4;
27
28 // Assign output variables to GPIO pins
29 const int Led_00 = 26;
30 const int Led_01 = 27;
31 const int Led_02 = 14;
32 const int Led_03 = 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     RB = Register_bank(ACC.O, Adr, Clk, Wr_RB);
50
51     ALU4 = ALU_4(ACC.O, RB.O, op);
52
53     MUX = MUX21(sel, I, ALU4.O);
54
55     ACC = Accumulator(MUX.O, Clk, Wr_ACC);
56
57     // LED outputs
58     digitalWrite(Led_00, ACC.O[0]);
59     digitalWrite(Led_01, ACC.O[1]);
60     digitalWrite(Led_02, ACC.O[2]);
61     digitalWrite(Led_03, ACC.O[3]);
62
63     Reg3.Q[0]=0;
64     Reg3.Q[1]=0;
65     Reg3.Q[2]=0;
66     Reg3.Q[3]=0;
67
68     Reg3.O[0]=0;
69     Reg3.O[1]=0;
70     Reg3.O[2]=0;
71     Reg3.O[3]=0;
72
73     ACC.Q[0] = 0;
74     ACC.Q[1] = 0;
75     ACC.Q[2] = 0;
76     ACC.Q[3] = 0;
77
78     ACC.O[0] = 0;
79     ACC.O[1] = 0;
80     ACC.O[2] = 0;
81     ACC.O[3] = 0;
82
83     // Initialize the LED variables as outputs
84     pinMode(Led_00,OUTPUT);
85     pinMode(Led_01,OUTPUT);
86     pinMode(Led_02,OUTPUT);
87     pinMode(Led_03,OUTPUT);
88
89     // Set LED outputs to LOW
90     digitalWrite(Led_00, LOW);
91     digitalWrite(Led_01, LOW);
92     digitalWrite(Led_02, LOW);
93     digitalWrite(Led_03, LOW);
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112

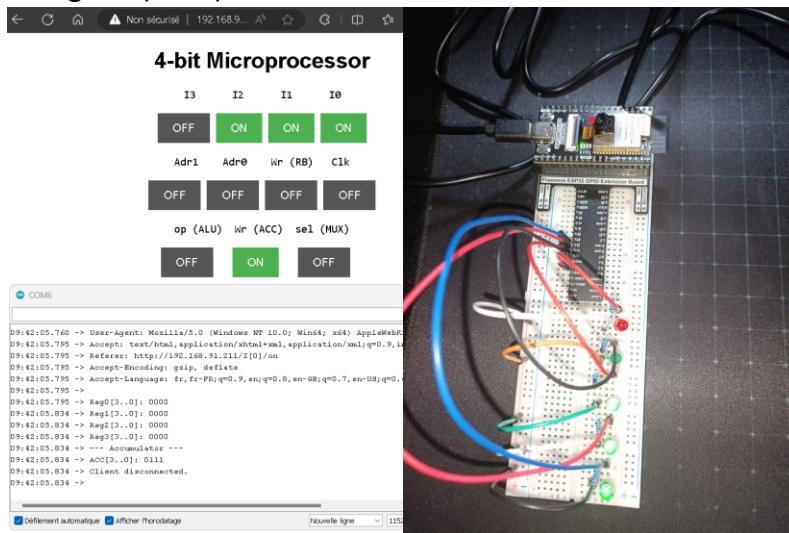
```

EX.1.4 Programmation :

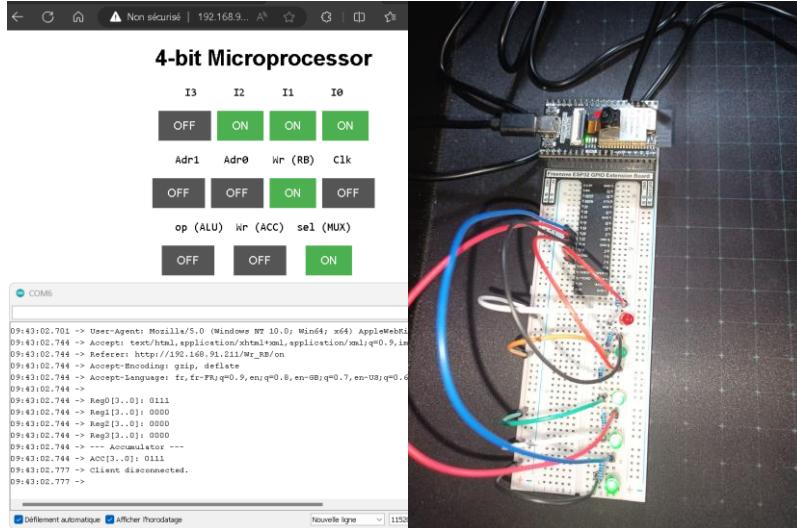
Expliquer également comment vous procédez cette opération :

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

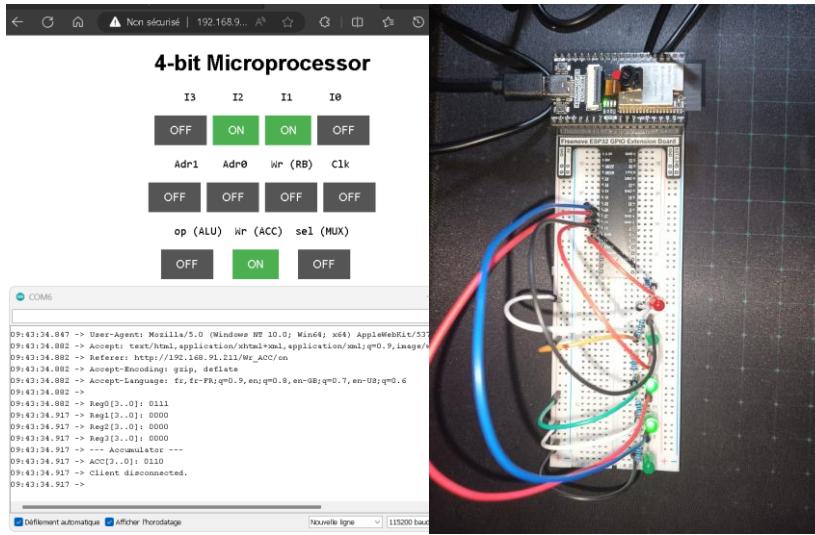
1. Charger 7 (0111) dans l'accumulateur :



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

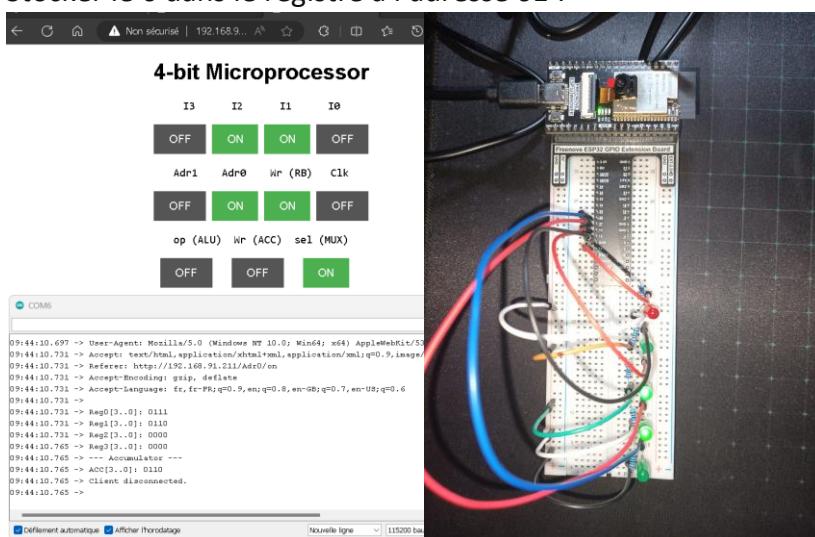


- 4.
5. Charger 6 (0110) dans l'accumulateur :



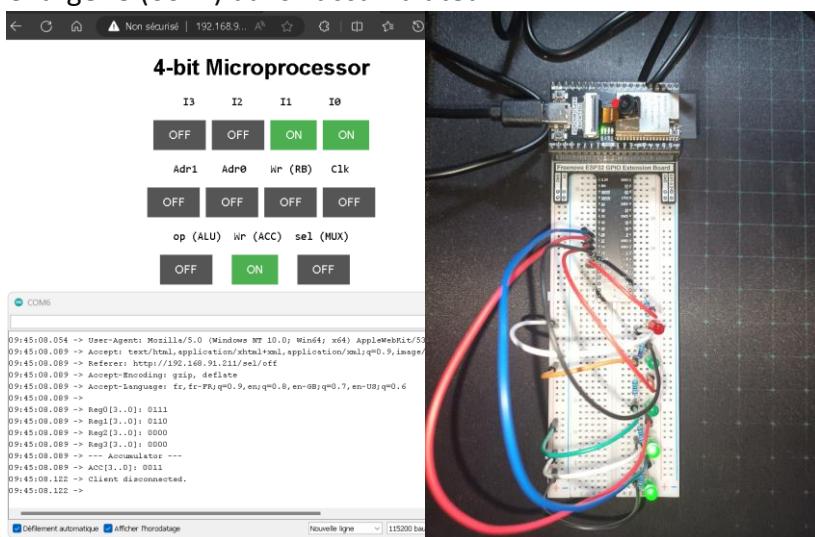
6.

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



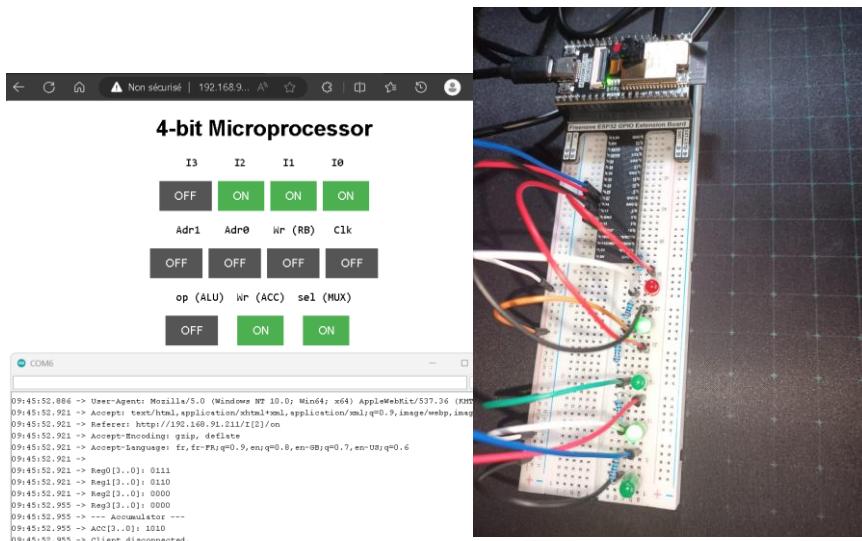
8.

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



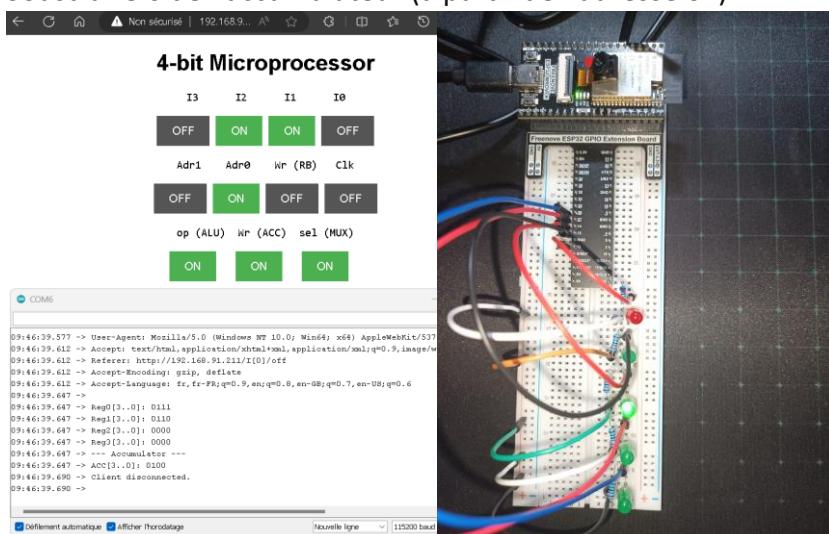
10.

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) :



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 :

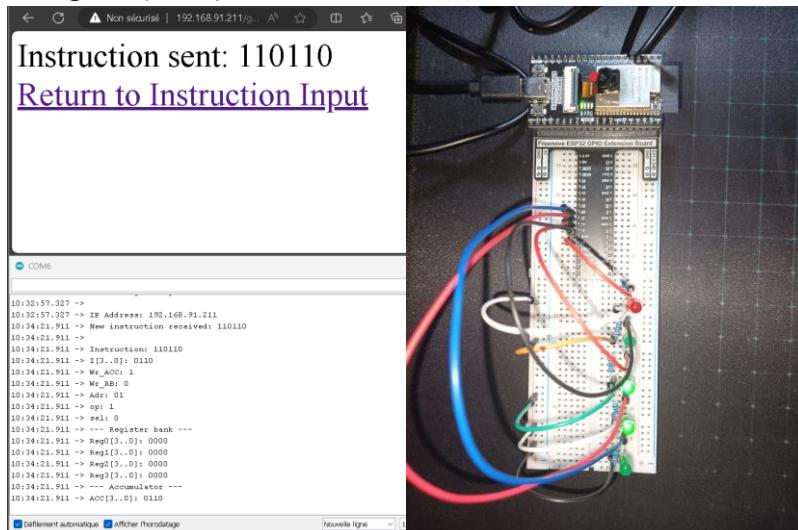
```
1 #include <WiFi.h>
2 #include <AsyncTCP.h>
3 #include <ESPAsyncWebServer.h>
4 #include "circuits.h"
5
12 const char* PARAM_INPUT = "Instruction";
13 char instruction[7];
14
15
16 // Declaration of the variables of the Micro-processor
17 bool Wr_ACC, I[4], op, Wr_RB, Adr[2], sel, Clk;
18
19 Reg Reg0, 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;
```

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 :

```
Instruction sent: 100000
Return to Instruction Input
```

- 4.

5. Charger 3 (0011) dans l'accumulateur :

```
Instruction sent: 110011
Return to Instruction Input
```

- 6.

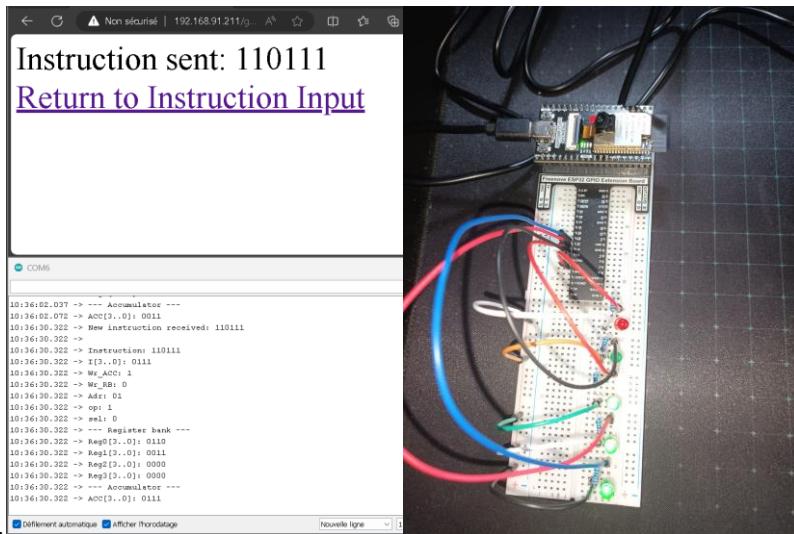
7. Stocker le 3 dans le registre à l'adresse 01 :

```
Instruction sent: 100100
Return to Instruction Input
```

- 8.

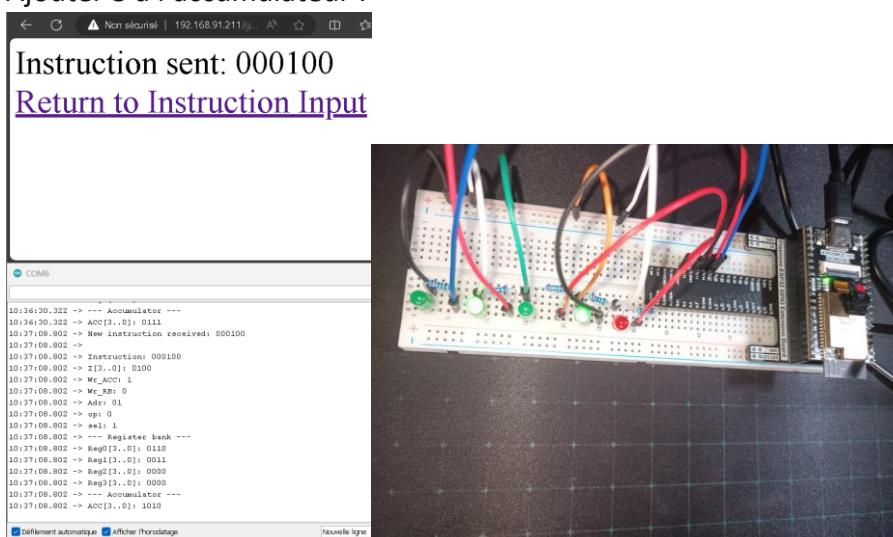
9. Charger 7 (0111) dans l'accumulateur :

```
Instruction sent: 100111
Return to Instruction Input
```



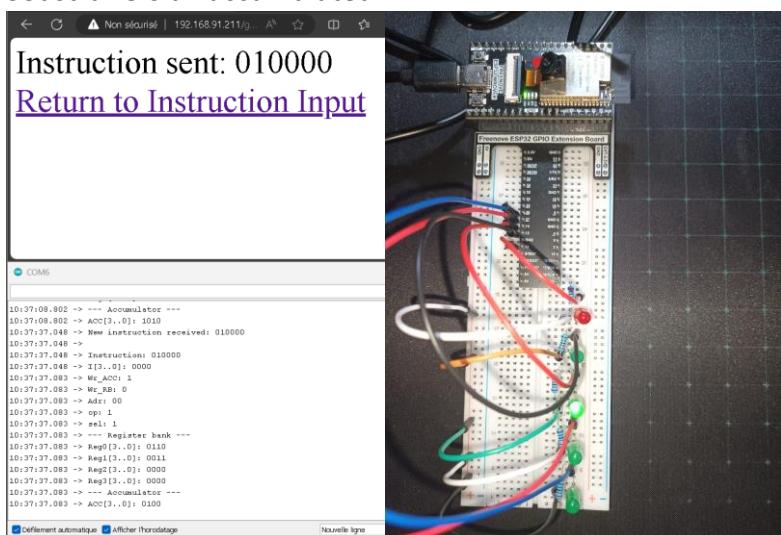
10.

11. Ajouter 3 à l'accumulateur :



12.

13. Soustraire 6 à l'accumulateur :

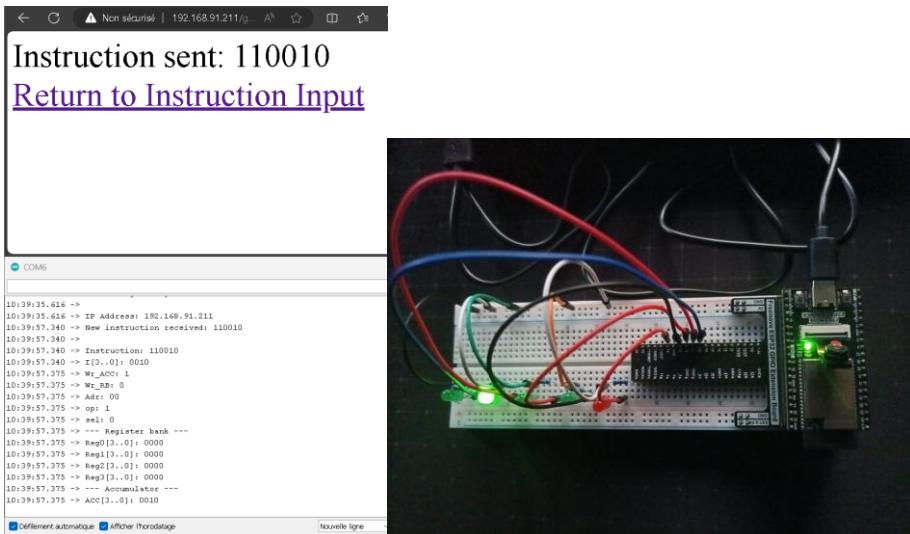


14.

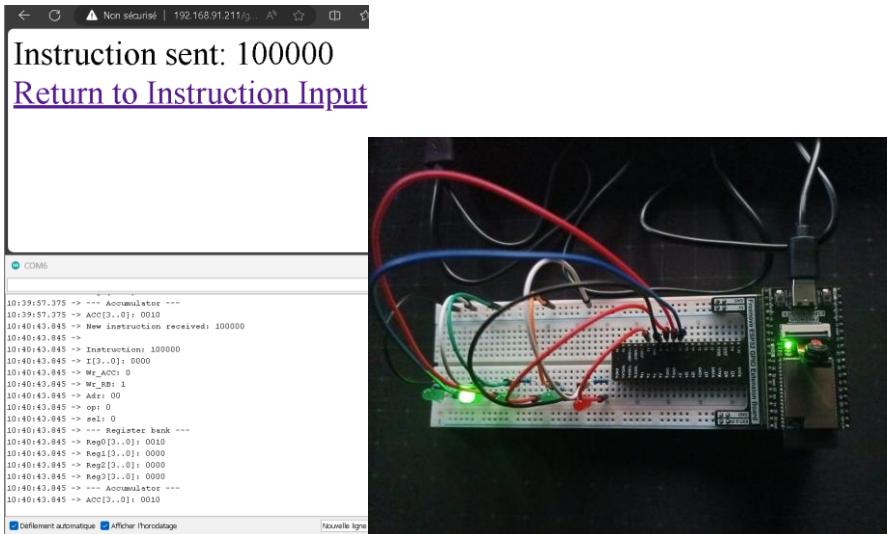
Instruction binaire 7+3-6+5-2 :

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

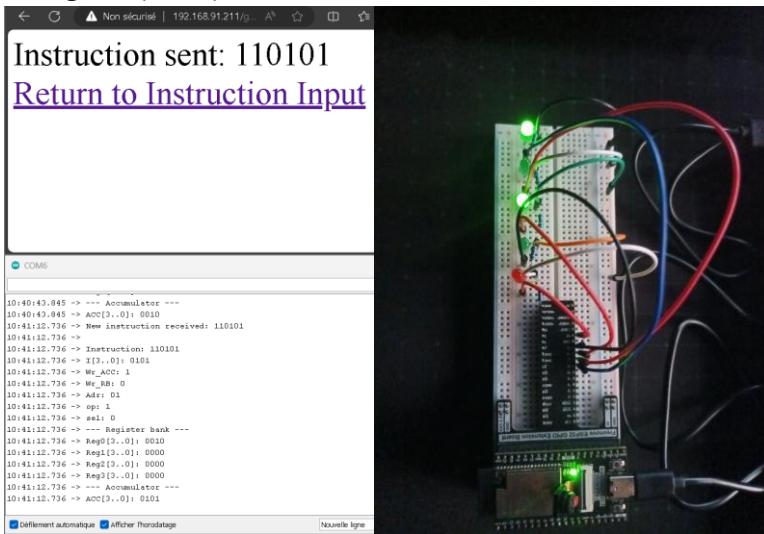
15. Charger 2 (0010) dans l'accumulateur :



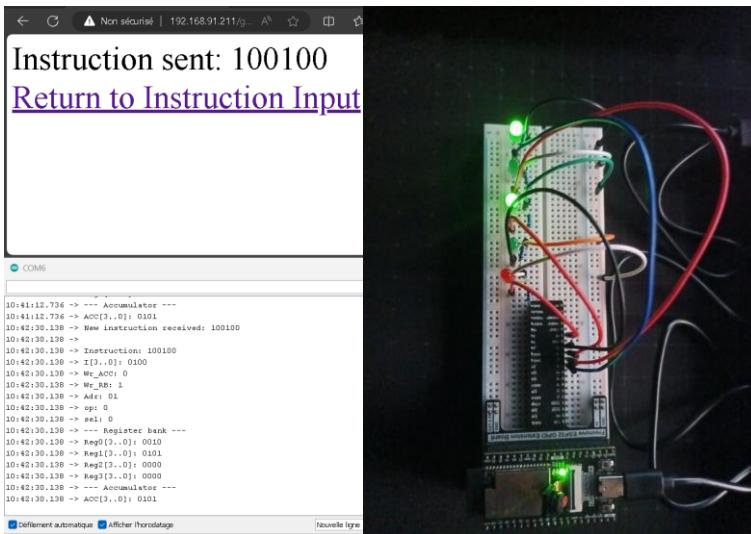
16.
17. Stocker le 2 dans le registre à l'adresse 00 :



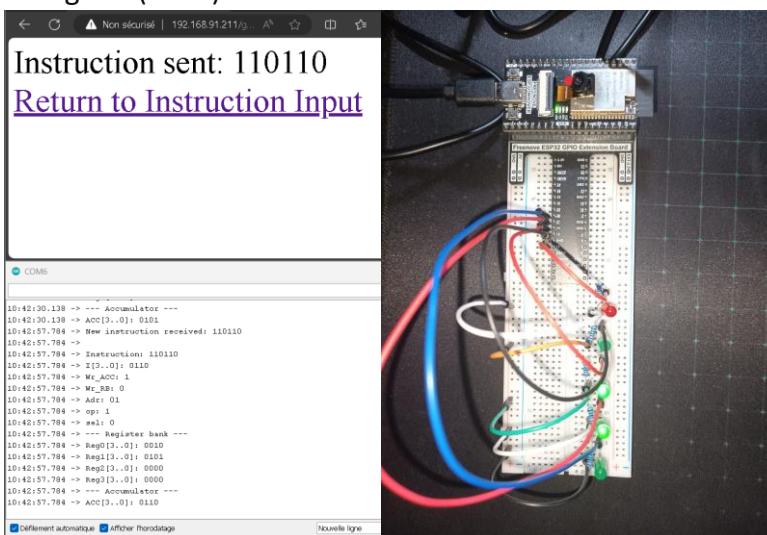
18.
19. Charger 5 (0101) dans l'accumulateur :



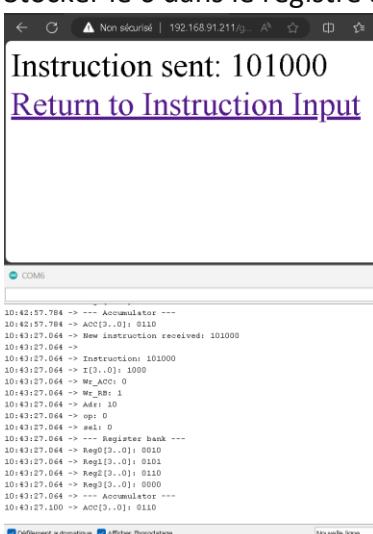
20.
21. Stocker le 5 dans le registre à l'adresse 01 :



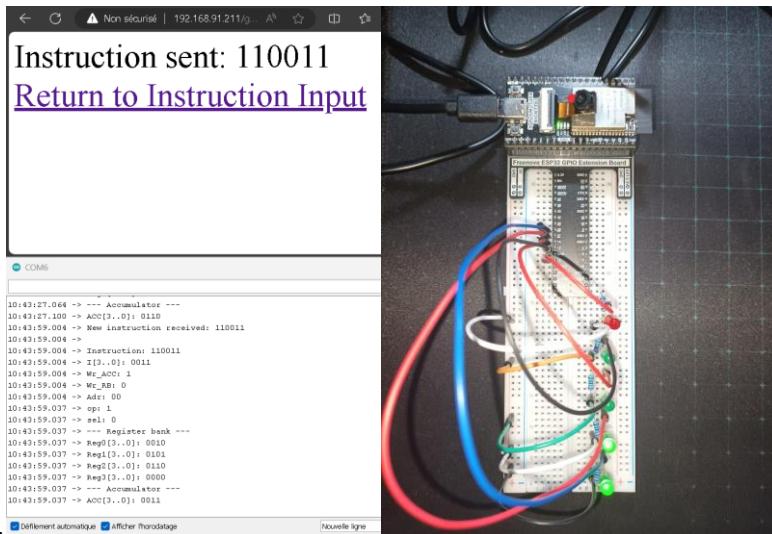
23. Charger 6 (0110) dans l'accumulateur :



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

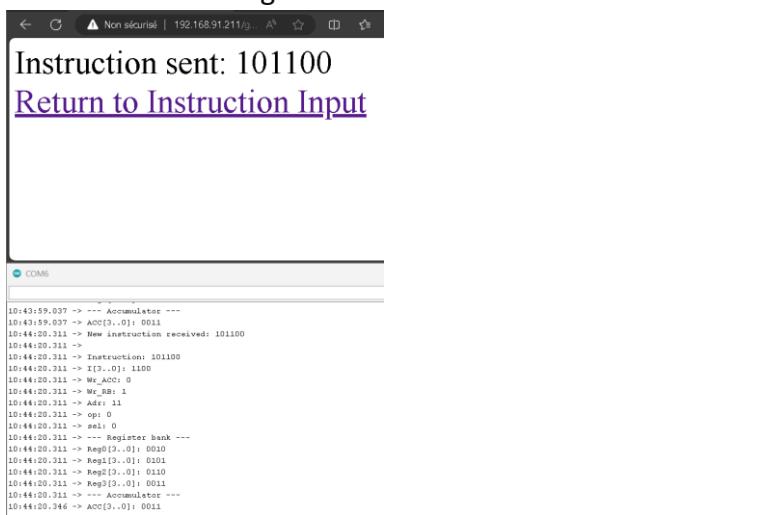


27. Charger 3 (0110) dans l'accumulateur :



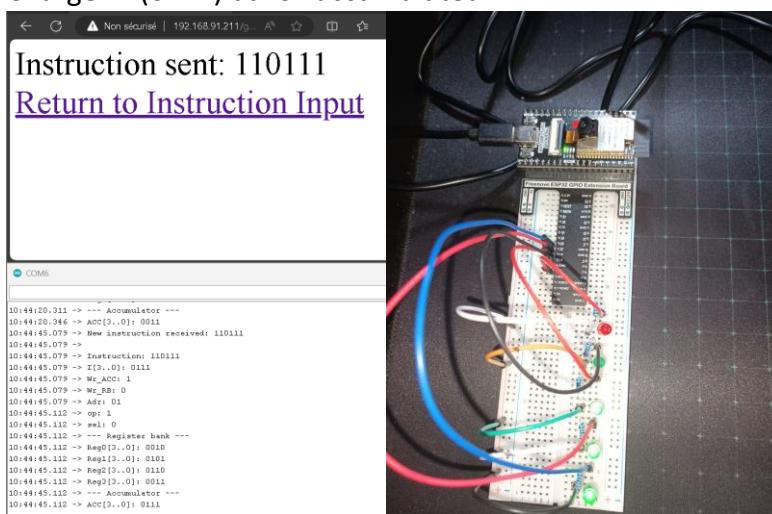
28.

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



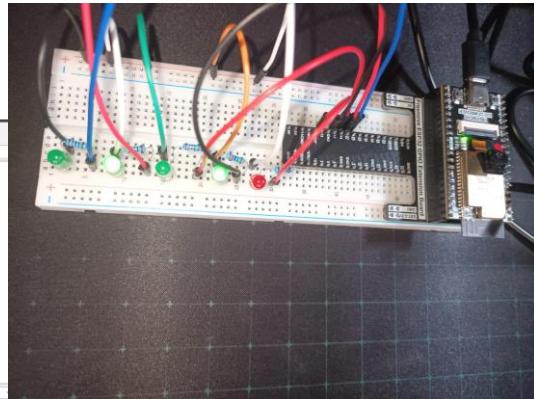
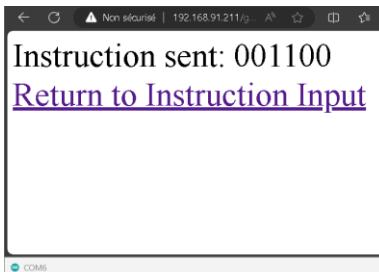
30.

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



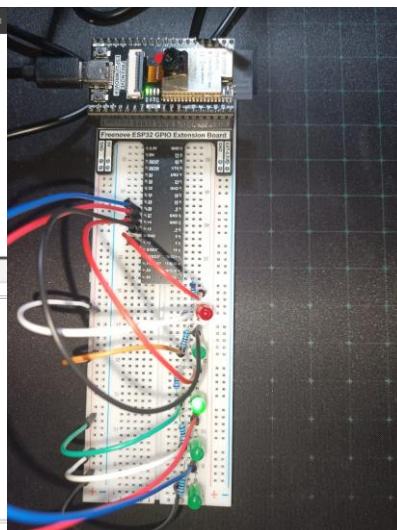
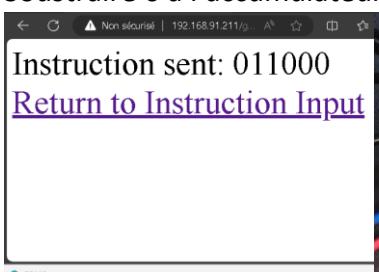
32.

33. Ajouter 3 à l'accumulateur :



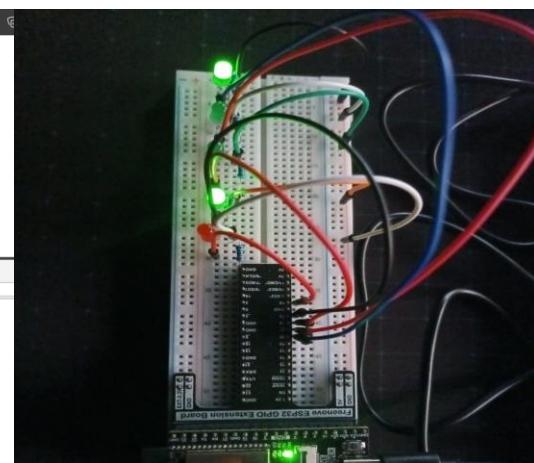
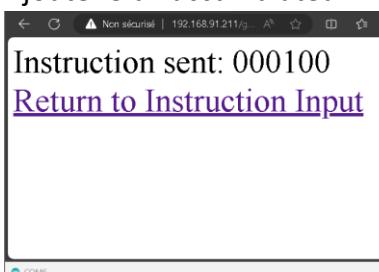
34.

35. Soustraire 6 à l'accumulateur :



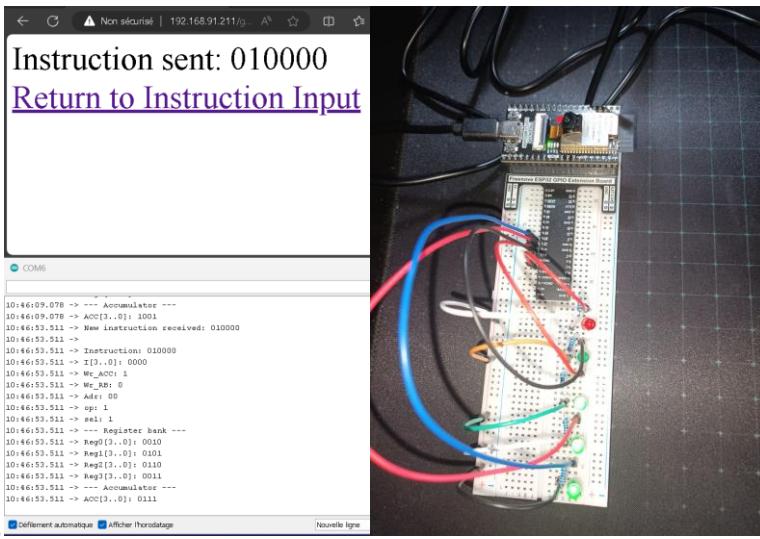
36.

37. Ajouter 5 à l'accumulateur :



38.

39. Soustraire 2 a l'accumulateur :



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