***NOLLE***

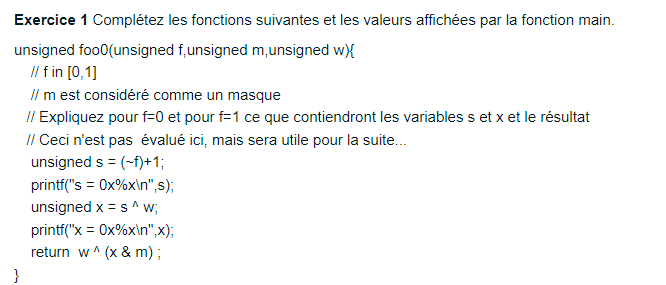
***Damien***

***L3 – Informatique***

**ADO – Devoir 1 :**

|  |  |
| --- | --- |
| ***Note :*** | ***Observation :*** |
| ***/20*** |  |

Exercice 1)



Tous les paramètres de la fonction sont des unsigned. On est donc sur des nombres entiers non signés.

s = (~f) + 1 🡪 On fait ici le complément à 2 : NOT(F) + 1, il s’agit donc de la représentation binaire négatif de f.

printf("s = 0x%x\n",s); 🡪 On affiche le résultat de l’opération précédente en hexadécimal.

unsigned x = s ^ w; 🡪 Nous déclarons une variable unsigned (non signé, soit strictement positif) et

nous lui affectons le résultat d’un XOR (ou exclusif) entre s et w.

printf("x = 0x%x\n",x); 🡪 Nous affichons le contenu de la variable x, soit le résultat de l’opération

précédente.

return w ^ (x & m) ; 🡪 Nous retournons le résultat d’un XOR entre w et le résultat d’un AND (et

logique) entre x et m.

w = 222 = 0b1101 1110

222 = 111 \* 2 + 0

111 = 55 \* 2 + 1

55 = 27 \* 2 + 1

27 = 13 \* 2 + 1

13 = 6 \* 2 + 1

6 = 3 \* 2 + 0

3 = 1 \* 2 + 1

1 = 0 \* 2 + 1

m = 111 = 0b0110 1111

111 = 55 \* 2 + 1

55 = 27 \* 2 + 1

27 = 13 \* 2 + 1

13 = 6 \* 2 + 1

6 = 3 \* 2 + 0

3 = 1 \* 2 + 1

1 = 0 \* 2 + 1

* Pour f = 0 :

s = NOT(0) + 1 = 1111 1111 1111 1111 1111 1111 1111 1111 + 1 = 0

x = s XOR w = 0000 0000 0000 0000 0000 0000 0000 0000 XOR 1101 1110 = 1101 1110 = 222

w XOR (x AND m)

1101 1110

AND 0110 1111

0100 1110 = 64 + 8 + 4 + 2 = 72 + 6 = 78

1101 1110

XOR 0100 1110

1001 0000 = 128 + 16 = 144

* Pour f = 1 :

s = NOT(1) + 1 = 1111 1111 1111 1111 1111 1111 1111 1110 + 1 = 1111 1111 1111 1111 1111 1111 1111 1111 🡪 C’est la représentation binaire de -1.

x = s XOR w = 1111 1111 1111 1111 1111 1111 1111 1111 XOR 1101 1110 = 1111 1111 1111 1111 1111 1111 0010 0001 🡪 C’est la représentation binaire de - 222 (-255 + 33 = -222)

w XOR (x AND m)

1111 1111 1111 1111 1111 1111 0010 0001

AND 0000 0000 0000 0000 0000 0000 0110 1111

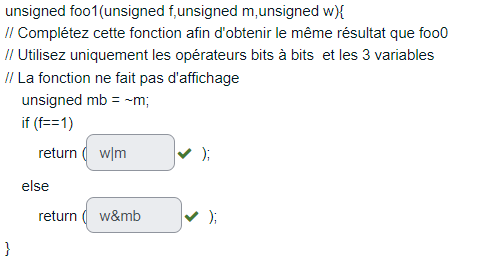
0000 0000 0000 0000 0000 0000 0010 0001 = 1 + 32 = 33

1101 1110

XOR 0010 0001

1111 1111 = 255

Les opérations pour s et x permettent d’obtenir la représentation binaire négative d’un nombre si f = 1, positive si f = 0.



mb = NOT(m) = NOT(0000 0000 0000 0000 0000 0000 0110 1111) = 1111 1111 1111 1111 1111 1111 1001 0000

f = 1

x = ~f + 1 ^ w

Réponse : w|m

car :

1101 1110

| 0110 1111

1111 1111

w = 161 = 1010 0001

m = 54 = 0011 0110

161 – 128 = 33 – 32 = 1 – 1 = 0

54 -32 = 22 – 16 = 6 – 4 = 2 – 2 = 0

s = NOT(1) + 1 = 1111 1111 1111 1111 1111 1111 1111 1110 + 1 = 1111 1111 1111 1111 1111 1111 1111 1111

x = 1111 1111 1111 1111 1111 1111 1111 1111 XOR 1010 0001

= 1111 1111 1111 1111 1111 1111 0101 1110

1010 0001 XOR (1111 1111 1111 1111 1111 1111 0101 1110 AND 0011 0110)

= 1010 0001 XOR 0001 0110 = 1011 0111

w|m = 1010 0001 | 0011 0110 = 1011 0111

f = 0

x = w

Réponse : w & mb

car :

w = 0000 0000 0000 0000 0000 0000 1101 1110

mb = 1111 1111 1111 1111 1111 1111 1001 0000

0000 0000 0000 0000 0000 0000 1101 1110

& 1111 1111 1111 1111 1111 1111 1001 0000

0000 0000 0000 0000 0000 0000 1001 0000

m = 189 = 1011 1101

w = 164 = 1010 0100

189 – 128 = 61 – 32 = 29 – 16 = 13 – 8 = 5 = 4 = 1 – 1 = 0

164 – 128 = 36 – 32 = 4 – 1 = 0

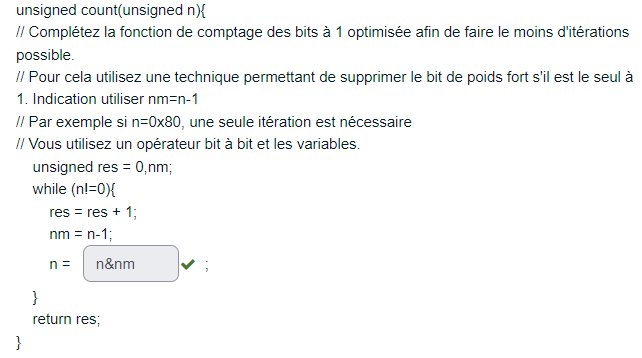
x = w = 1010 0100

w XOR (x AND m) = 1010 0100 XOR (1010 0100 AND 1011 1101)

= 1010 0100 XOR 1010 0100 = 0000 0000 = 0

mb = NOT(m) = NOT(1011 1101) = 0100 0010

w & mb = 1010 0100 & 0100 0010 = 0000 0000 = 0



Réponse : n = n & nm

n = 128 = 1000 0000

1ère itération :

nm = 127 = 0111 1111

n = 1000 0000 & 0111 1111 = 0000 0000 = 0

n = 135 = 1000 0111

1ère itération :

nm = 134 = 1000 0110

n = 1000 0111 & 1000 0110 = 1000 0110

2ème itération :

nm = 133 = 1000 0101

n = 1000 0110 & 1000 0101 = 1000 0100 = 128 + 4 = 132

3ème itération :

nm = 131 = 1000 0011

n = 1000 0100 & 1000 0011 = 1000 0000 = 128

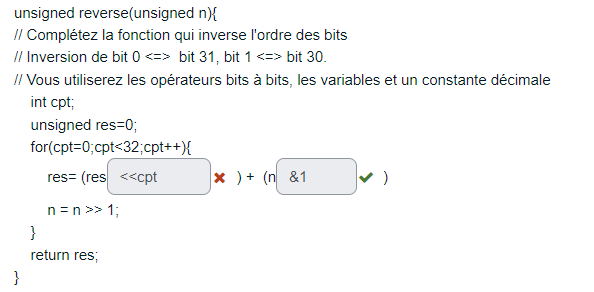
4ème itération :

nm = 127 = 0111 1111

n = 1000 0000 & 0111 1111 = 0000 0000

Nous constatons qu’avec cette méthode, nous prenons moins d’itération que la version de base de la fonction permettant de compter les bits à 1, qui consiste en un SRL 1 avec la valeur n à chaque itération et res = n & 1, ce qui fait passer en revue tous les bits de n.

Dans la version optimiser, le nombre d’itération correspond au nombre de bits à 1.



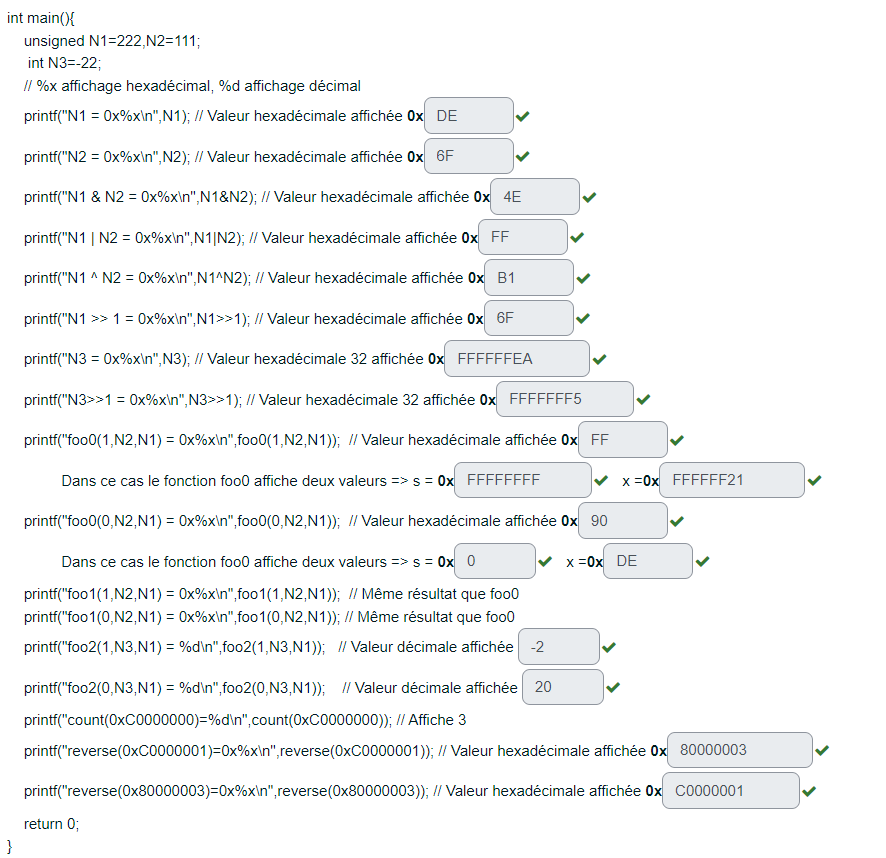
Réponse : (res << cpt) + (n & 1)

car cpt = 0, donc res ne fera aucun décalage et on y ajoutera le bit tout à droite.

On fait ensuite un décalage à droite de 1 à n.

Puis lorsque cpt = 1, alors on fait un décalage à gauche de 1 à res et on y placera le bit le plus à droite.

Et ainsi de suite…



N1 = 222 = 1101 1110

N2 = 111 = 0110 1111

N3 = -22

N1 en hexadécimal :

13 🡪 D

14 🡪 E

222 = 13 \* 16 + 14

13 = 0 \* 16 + 13

Réponse : 0xDE

N2 en hexadécimal :

6

15 🡪 F

111 = 6 \* 16 + 15

6 = 0 \* 16 + 6

Réponse : 0x6F

N1 & N2 :

1101 1110

& 0110 1111

0100 1110

4

14 🡪 E

Réponse : 0x4E

N1 | N2 :

1101 1110

| 0110 1111

1111 1111

15 🡪 F

Réponse : 0xFF

N1 ^ (XOR) N2

1101 1110

^ 0110 1111

1011 0001

11 🡪 B

1

Réponse : 0xB1

N1 >> (SRL) 1

1101 1110 >> 1 = 0110 1111

Réponse : 0x6F

N3 en hexadécimal :

22 = 0000 0000 0000 0000 0000 0000 0001 0110

-22 = ~(22) + 1 (Complément à 2) =

1111 1111 1111 1111 1111 1111 1110 1001 + 1 = 1111 1111 1111 1111 1111 1111 1110 1010

Réponse : 0xFFFFFFEA

14 🡪 E

10 🡪 A

N3 >> 1 :

Attention :

int 🡪 SRA

unsigned 🡪 SLL

1111 1111 1111 1111 1111 1111 1110 1010 >> 1 = 1111 1111 1111 1111 1111 1111 1111 0101

Réponse : 0xFFFFFFF5

f = 0

w = 222 = 1101 1110

s = NOT(0) + 1 = 1111 1111 1111 1111 1111 1111 1111 1111 + 1 = 0

x = s XOR w = 0000 0000 0000 0000 0000 0000 0000 0000 XOR 1101 1110 = 1101 1110 = 222

m = 111 = 0110 1111

w XOR (x AND m) = 1101 1110 XOR (1101 1110 AND 0110 1111) = 1101 1110 XOR 0100 1110 = 1001 0000 = 90

f = 1

w = 222 = 1101 1110

s = NOT(1) + 1 = 1111 1111 1111 1111 1111 1111 1111 1110 + 1 = 1111 1111 1111 1111 1111 1111 1111 1111 🡪 C’est la représentation binaire de -1.

x = s XOR w = 1111 1111 1111 1111 1111 1111 1111 1111 XOR 1101 1110 = 1111 1111 1111 1111 1111 1111 0010 0001 🡪 C’est la représentation binaire de - 222 (-255 + 33 = -222) = FFFFFF21

m = 111 = 0110 1111

w XOR (x AND m) = 1101 1110 XOR (1111 1111 1111 1111 1111 1111 0010 0001 AND 0110 1111) = 1101 1110 XOR 0010 0001 = 1111 1111

int foo2(int f,int m,int w){

// Même fonction que foo0 sur les nombres entiers signés

int x = (-f) ^ w;

return w ^ (x & m) ;

}

f = 1

x = -1 XOR 222

x = 1111 1111 1111 1111 1111 1111 1111 1111 XOR 1101 1110

x = 1111 1111 1111 1111 1111 1111 0010 0001

1101 1110 XOR (1111 1111 1111 1111 1111 1111 0010 0001 AND 1111 1111 1111 1111 1111 1111 1110 1010)

= 1101 1110 XOR 1111 1111 1111 1111 1111 1111 0010 0000

= 1111 1111 1111 1111 1111 1111 1111 1110

1111 1111 1111 1111 1111 1111 0010 0001

AND 1111 1111 1111 1111 1111 1111 1110 1010

1111 1111 1111 1111 1111 1111 0010 0000

Réponse : -1 – 1 = -2

f = 0

x = -0 XOR 222

x = 0000 0000 XOR 1101 1110

x = 1101 1110

1101 1110 XOR (1101 1110 AND 1111 1111 1111 1111 1111 1111 1110 1010)

= 1101 1110 XOR 1100 1010

= 0001 0100

0000 0000 0000 0000 0000 0000 1101 1110

AND 1111 1111 1111 1111 1111 1111 1110 1010

0000 0000 0000 0000 0000 0000 1100 1010

Réponse : 16 + 4 = 20

reverse(0xC0000001)

C 🡪 12 = 1100

1 = 0001

1100 0000 0000 0000 0000 0000 0000 0001

Bit 0 🡪 bit 31 et bit 31 🡪 bit 0

Etc…

Réponse : 1000 0000 0000 0000 0000 0000 0000 0011 = 0x80000003

Reverse(80000003)

1000 0000 0000 0000 0000 0000 0000 0011

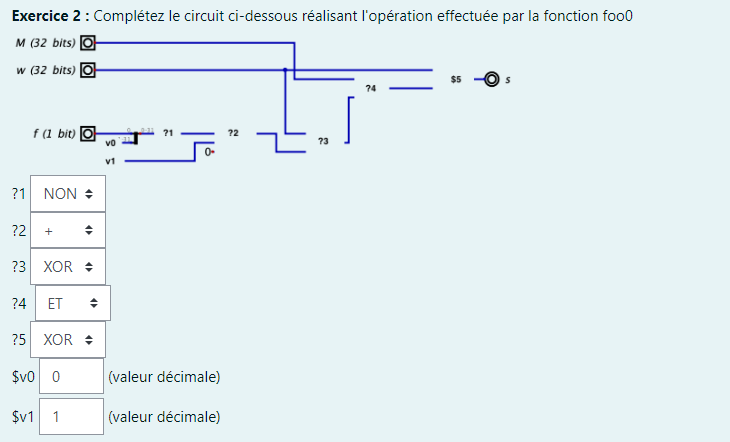
Bit 0 🡪 bit 31 et bit 31 🡪 bit 0

Etc…

Réponse : 1100 0000 0000 0000 0000 0000 0000 0001 = C0000001

12 🡪 C

Exercice 2)



Il suffisait de lire le code de la fonction foo0 :

s = (~f) + 1 ; 🡪 NOT(f) + 1

unsigned x = s ^ w; 🡪 s XOR w

return w ^ (x & m) ; 🡪 w XOR (x AND m)