

Rapport du projet :

***Conversion d'un nombre signé en
représentation de valeur signée (VS),
complément à un (Cà1) et complément à
deux (Cà2)***

Réalisé par : **Bilal ZINEDDINE** et **Yasmine HAMDANI**

Encadre par: Mr. **BENALLA Hicham**

Introduction :

La conversion entre différentes représentations de nombres signés est essentielle dans l'informatique, en particulier lorsque l'on travaille avec des processeurs et des systèmes qui utilisent des formats de données spécifiques. Ce rapport présente un projet visant à résoudre le problème de la conversion de nombres signés en trois représentations différentes : la valeur signée (VS), le complément à un et le complément à deux. Les opérations de conversion sont importants pour garantir la précision des calculs et la gestion des valeurs négatives.

Analyse du Problème :

Le projet vise à résoudre le problème de la conversion de nombres signés en différentes représentations, à **savoir** la valeur signée (VS), le complément à un (Ca1) et le complément à deux (Ca2). On va traiter les cas des nombre positive et les nombres négatifs , les problemes de debordement et les methodes utilise dans chaque conversion .

Solutions Proposées :

Les solutions pour la conversion entre les représentations VS, Ca1 et Ca2, ainsi que la conversion entre ces représentations et des valeurs entières, voici l'algorithme a suivre pour elaborer ce genre de solution :

Plan d'algorithme :

Entrée :

L'utilisateur est invité à entrer le nombre de bits dans laquelle on va le représenter le nombre signe ,puis il se derige vers un menu qui lui offre des differentes conversion entre les differentes representation du nombre signe et l'inverse.

Probleme de signe :

Dans la conversion d'un nombre signé en representation quelconque il faut rectifier le sign , si il'est positif donc les representation reste normale que dans le binaire pure, tandis s'il est negatif il faut faire attention au bit de signe qui est le bit du poid fort.

Probleme de debordement :

Le probleme de debordement est lie a l'intervalle convenable qu'on peut effectuer la conversion , il a une relation avec le nombre de bits donne pour la representation , on definit les trois intervalle pour chaque representation :

VS : $[-2^{nbBits} ; 2^{nbBits}]$

Cà1 : $[1 - 2^{nbBits} ; -1 + 2^{nbBits}]$

Cà2 : $[-2^{nbBits} ; -1 + 2^{nbBits}]$

Conversion de nombre signe en representation VS :

Pour la conversion de nombre signe en valeur signe on prends la representation en binaire pure de son valeur absolue , puis on ajout le bit de signe en bit du poids fort.

Conversion de nombre signe en representation cà1 :

Pour arriver au complment à 1 , on va juste inverser(les 1 par 0 et 0 par 1) tous les bits sauf le bit de signe.

Conversion de nombre signe en representation cà2 :

Enfin, pour atteindre au complement à 2 , il faut ajouter un 1 a la representaion de complement à 1.

Le programme traduit en langage C :

1. 3 Fonctions qui teste le debordement dans chaque cas de representaion :

```
bool testDebordementC2(int n, int nbBits) {
    int intervalle_min = -pow(2, nbBits) / 2;
    int intervalle_max = -1 + pow(2, nbBits) / 2;

    if (n >= intervalle_min && n <= intervalle_max) {
        return true;
    } else {
        return false;
    }
}

bool testDebordementC1(int n, int nbBits) {
    int intervalle_min = 1 - pow(2, nbBits) / 2;
    int intervalle_max = -1 + pow(2, nbBits) / 2;

    if (n >= intervalle_min && n <= intervalle_max) {
        return true;
    } else {
        return false;
    }
}

bool testDebordementVS(int n, int nbBits) {
    int intervalle_min = -pow(2, nbBits);
    int intervalle_max = pow(2, nbBits);

    if (n >= intervalle_min && n <= intervalle_max) {
        return true;
    } else {
        return false;
    }
}
```

Les fonction ci-dessus sert a tester s'il y a un debordement pour les trois cas , en calculant l'intervalle de nombre signé qui peuvent représenter sur le nombre de bits demandé, en prenant le nombre signe et le nombre de bits autant qu'arguments puis ils retournent vrai si il y a pas de debordement et faux sinon.

2. Fonction pour afficher un nombre en binaire :

```
void affiche(int* tab, int taille) {  
    for(int i=0; i<taille; i++)  
        printf("%d", tab[i]);  
}
```

La fonction *affiche(int* tab, int taille)*, affiche un nombre binaire sous forme d'un tableau.

3. Fonction pour calculer le nombre de chiffre d'un nombre :

```
int nbr_digits_dec(int x) {  
    int digit=0;  
    int A=x;  
    while (A!=0) {  
        A/=2;  
        digit++;  
    }  
    return digit;  
}
```

La fonction *nbr_digits_dec(int x)* détermine combien de chiffres sont nécessaires pour représenter un nombre entier en format binaire. Elle répète une succession de division par 2.

4. Fonction qui calcule nombre de bits d'un nombre binaire :

```
int nbr_digits_bin(double y) {  
    int digit = 0;  
    double A = y;  
    while(A >= 1) {  
        A /= 10;  
        digit++;  
    }  
    return digit;  
}
```

La fonction *nbr_digits_bin(double y)* Calcule le nombre de chiffres qu'un nombre binaire aurait s'il était écrit en décimal. Elle divise le nombre binaire par 10 jusqu'à ce qu'il soit inférieur à 1.

5. Fonction pour convertir un nombre binaire en décimal :

```
int* tbBin(double bin) {  
    int taille = nbr_digits_bin(bin);  
    int* binaryArray = (int*)malloc(taille * sizeof(int));  
    int B = (int)bin;  
    int index = 0;  
    while(B != 0) {  
        binaryArray[taille - index - 1] = B % 10;  
        B /= 10;  
        index++;  
    }  
    return binaryArray;  
}
```

La fonction *tbBin(double bin)* convertit un nombre binaire en un tableau de chiffres. La taille du tableau est définie par la fonction *nbr_digits_bin*, et les chiffres du nombre binaire sont insérés dans le tableau.

6. Fonction convertit un nombre binaire en decimale :

```
41 int binEnDec(int* presBin,int taille) {  
42     int resultat = 0;  
43     for (int i = 0; i < taille; i++) {  
44         resultat += presBin[i] * (int)pow(2, taille - 1 - i);  
45     }  
46     return resultat;  
47 }
```

La fonction *binEnDec(int* presBin,int taille)* transforme un tableau représentant un nombre binaire en sa valeur décimale. Elle multiplie chaque bit par 2 à les puissances de son poids correspondant et les additionne ensemble.

7. Fonction qui convertit un nombre decimale en un nombre binaire :

```
85 int* decEnBin(int decimalNumber, int nbrBits) {  
86     int quotient = decimalNumber;  
87     int* binaryArray = (int*)malloc(nbrBits * sizeof(int));  
88  
89     // Initialisation du tableau avec des zéros  
90     for (int i = 0; i < nbrBits; i++) {  
91         binaryArray[i] = 0;  
92     }  
93  
94     int index = 0;  
95  
96     // Convertir en binaire et stocker dans le tableau  
97     while (quotient != 0 && index < nbrBits) {  
98         binaryArray[nbrBits - index - 1] = quotient % 2;  
99         quotient /= 2;  
100        index++;  
101    }  
102  
103    return binaryArray;  
104 }  
105
```

La fonction *int* decEnBin(int decimalNumber, int nbrBits)* prend un nombre entier décimal et le nombre de bits désirés en tant que paramètres, si le nombre est zéro, le tableau est initialisé avec des zéros. Sinon, elle calcule la représentation binaire en divisant le nombre par deux et en stockant les restes (0 ou 1) dans le tableau. Ce processus continue jusqu'à ce que le nombre soit entièrement converti ou que la limite de bits soit atteinte. Les positions restantes dans le tableau sont remplies de zéros si nécessaire.

8. Fonction pour convertir un nombre signé en representation valeur signée VS :

```
107 // Fonction pour convertir un nombre en nombre signee (VS)
108 int* decEnVs(int n, int nbBits) {
109
110     int* resultat;
111     if (n >= 0) {
112         resultat = decEnBin(n, nbBits);
113     } else {
114         resultat = decEnBin(-n, nbBits);
115         resultat[0] = 1;
116     }
117     return resultat;
118 }
119
```

La fonction *decEnVs(int n, int nbBits)* prend un nombre décimal signe et le nombre de bits autant qu'arguments, on teste toujours le signe du nombre, si il est positif donc on va juste la convertit en binaire pure, si il est negative on code sa valeur absolue puis on ajoute le bits de signe en bit de poids fort .

9. Fonction pour convertir la representation valeur signée VS en un nombre signé:

```
120 // Fonction pour convertir une nombre signee (VS) en representation binaire pure
121 int* vsEnDec(int* presBin, int nbrBits) {
122     if (presBin[0] == 1) {
123         // Si le bit de signe est 1, c'est un nombre negatif.
124         // Pour le convertir en binaire pur, changez le bit de signe en 0.
125         presBin[0] = 0;
126         return -binEnDec(presBin, nbrBits);
127     }
128     return binEnDec(presBin, nbrBits);
129 }
130
```

La fonction *vsEnDec(int* presBin, int nbrBits)* prend une chaine de caractere (la presentaion en binaire) autant qu'arguments, toujours le test sur le signe de nombre s'il est positive on retourne sa valeur directement , sinon on change le bit de signe puis on fait appel au fonction *binEnDec* en retournant la valeur négative retourner par la fonction precedente puisqu'il s'agit d'un nombre negatif.

10. Fonction pour convertir un nombre signé en représentation complément à 1:

```
133 int* decEnC1(int n, int nbBits) {
134     if (n >= 0) {
135         return decEnBin(n, nbBits);
136     } else {
137         int* presBinVS = decEnVs(n, nbBits);
138
139         // Ignorer le bit de signe, qui est le premier bit (index 0)
140         for (int i = 1; i < nbBits; i++) {
141             if (presBinVS[i] == 0) {
142                 presBinVS[i] = 1;
143             } else {
144                 presBinVS[i] = 0;
145             }
146         }
147
148         return presBinVS;
149     }
150 }
151
```

La fonction `decEnC1(int n, int nbBits)` prend un nombre décimal signé et le nombre de bits autant qu'arguments, un test de signe se fait si il est positif on retourne directement sa représentation en binaire pure, sinon on fait appel à la fonction `decEnVs` qui va convertir notre nombre en représentation à valeur signée ensuite on fait une boucle pour qui va changer les bits de la représentation VS de 0 en 1 et 1 en 0, en ignorant le bit de signe.

11. Fonction pour convertir la représentation complément à 1 en un nombre signé:

```
153 int clEnDec(int* presBin, int nbrBits) {
154     if (presBin[0] == 1) {
155         int* resultat = presBin; // Allouer de la mémoire pour le résultat
156         // Inversion des bits sauf le bit de signe
157         for (int i = 1; i < nbrBits; i++) {
158             if (presBin[i] == 0) {
159                 resultat[i] = 1;
160             } else {
161                 resultat[i] = 0;
162             }
163         }
164         return vsEnDec(resultat, nbrBits);
165     } else {
166         // Sinon, c'est un nombre positif en Cal
167         return binEnDec(presBin, nbrBits);
168     }
169 }
```

La fonction `c1EnDec(int* presBin, int nbrBits)` convertit une représentation binaire en Complément à un (Ca1) en un nombre décimal signé. Si le premier bit (bit de signe) est 1, indiquant un nombre négatif, la fonction inverse les bits restants, à l'exception du bit de signe. Après cette inversion, elle appelle `vsEnDec` pour obtenir la valeur décimale signée correspondante. Si le bit de signe est 0, indiquant un nombre positif, elle appelle directement `binEnDec` pour convertir la représentation binaire en sa valeur décimale.

12. Fonction pour convertir un nombre signé en représentation complément à 2:

```

171 // Fonction pour convertir une representation binaire pure en complement a deux (Ca2)
172 int* decEnC2(int n, int nbBits) {
173     int* c1 = decEnC1(n, nbBits); // Convertir en Ca1
174     int* resultat = (int*)malloc(strlen(c1) + 1); // Allouer de la memoire pour le resultat
175     resultat = c1;
176     int retenue = 1;
177     for (int i = strlen(c1) - 1; i >= 0; i--) {
178         if (retenue == 0) {
179             resultat[i] = c1[i];
180         } else {
181             if (c1[i] == 0) {
182                 resultat[i] = 1;
183                 retenue = 0;
184             } else {
185                 resultat[i] = 0;
186             }
187         }
188     }
189     return resultat;
190 }
191

```

La fonction `decEnC2(int n, int nbBits)` convertit un nombre décimal signé en sa représentation en complément à deux (C2). Pour les nombres positifs, elle retourne directement leur représentation binaire. Pour les nombres négatifs, elle suit la formule : $\text{comp2}(N) = \text{comp1}(N) + 1$, où C1 est le complément à un du nombre. Elle utilise une méthode de retenue pour ajouter 1 au C1 : si un chiffre binaire est 0 et il y a une retenue, ce chiffre devient 1 et la retenue est annulée. Si le chiffre est 1, il devient 0 et la retenue est maintenue. Ce processus continue jusqu'à ce que tous les bits soient ajustés ou que la retenue soit nulle.

13. Fonction pour convertir la représentation complement à 2 en un nombre signé:

```
193 int c2EnDec(int* presBin, int nbBits) {
194     // Si le bit de signe est 1, c'est un nombre negatif
195     if (presBin[0] == 1) {
196         int* valC1 = presBin; // Allouer de la memoire pour le resultat
197         // Inversion des bits sauf le bit de signe
198         for (int i = 1; i < nbBits; i++) {
199             if (presBin[i] == 0) {
200                 valC1[i] = 1;
201             } else {
202                 valC1[i] = 0;
203             }
204         }
205
206         int* resultat = valC1; // Allouer de la memoire pour le resultat
207         int retenue = 1;
208         for (int i = nbBits - 1; i >= 0; i--) {
209             if (retenue == 0) {
210                 resultat[i] = valC1[i];
211             } else {
212                 if (valC1[i] == 0) {
213                     resultat[i] = 1;
214                     retenue = 0;
215                 } else {
216                     resultat[i] = 0;
217                 }
218             }
219         }
220         return vsEnDec(resultat, nbBits);
221     } else {
222         // Sinon, c'est un nombre positif en Ca2
223         return binEnDec(presBin, nbBits);
224     }
225 }
```

La fonction `c2EnDec(int* presBin, int nbBits)` prend une chaine de caractere (presentation binaire) autant qu'arguments, un test de signe se fait si il'est positive on retourne directement le nombre signe correspodant a l'aide du fonction `binEnDec` , sinon on se base sur l'idée de la relation suivante : $\text{comp2}(\text{comp2}(N))=N$, donc va faire le complement à 2 du notre presBin , puis on fait appel au fonction `vsEnDec` car le reultat obtenue apres le processus de complement a 2 dans notre fonction c'est qu'une chemin retour qui va nous ramener a la presentation en VS .

14. Fonction principale qui comporte un menu :

```
void convertirSigneMenu(int nbBits) {
    int choix ;
    printf("Choisissez l'operation :\n");
    printf("1. Conversion de nombre signe en representaion VS (Valeur signee)\n");
    printf("2. Conversion de representaion VS (Valeur signee) en nombre signe\n");
    printf("3. Conversion de de nombre signe en Ca1 (Complement a un)\n");
    printf("4. Conversion de Ca1 (Complement a un) en nombre signe\n");
    printf("5. Conversion de de nombre signe en Ca2 (Complement a deux)\n");
    printf("6. Conversion de Ca2 (Complement a deux) en nombre signe\n");
    printf("7. Quitter\n");
    scanf("%d", &choix );

    if (choix == 1) {
        int n;
        printf("Entrez un nombre entier : ");
        scanf("%d", &n);
        if (testDebordementVS(n, nbBits)) {
            affiche(decEnVs(n,nbBits),nbBits);
        } else {
            printf("Il y a un problème de debordement!\n");
        }
    } else if (choix == 2) {
        double nbr;
        printf("Entrez la representation binaire : ");
        scanf("%lf", &nbr);
        int *presBin=(int*)malloc(nbBits*sizeof(int));
        presBin= tbBin(nbr);
        printf("Votre nombre signe est : %d\n",vsEnDec(presBin,nbBits));
    } else if (choix == 3) {
        int n;
        printf("Entrez un nombre entier : ");
        scanf("%d", &n);
    }
```

```

256     } else if (choix == 3) {
257         int n;
258         printf("Entrez un nombre entier : ");
259         scanf("%d", &n);
260         if (testDebordementC1(n, nbBits)) {
261             printf("la representaion en Ca1 de %d est :\n", n);
262             affiche(decEnC1(n,nbBits),nbBits);
263         } else {
264             printf("Il y a un problème de débordement!\n");
265         }
266     } else if (choix == 4) {
267         double nbr;
268         printf("Entrez la representation Ca1 : ");
269         scanf("%lf", &nbr);
270         int *presBin=(int*)malloc(nbBits*sizeof(int));
271         presBin= tbBin(nbr);
272         printf("votre nombre signe est : %d\n", c1EnDec(presBin,nbBits));
273     } else if (choix == 5) {
274         int n;
275
276     } else if (choix == 5) {
277         int n;
278         printf("Entrez un nombre entier : ");
279         scanf("%d", &n);
280         if (testDebordementC2(n, nbBits)) {
281             printf("la representaion en Ca2 de %d est : \n", n);
282             affiche(decEnC2(n, nbBits),nbBits);
283         } else {
284             printf("Il y a un problème de débordement!\n");
285         }
286     } else if (choix == 6) {
287         double nbr;
288         printf("Entrez la representation Ca2 : ");
289         scanf("%lf", &nbr);
290         int *presBin=(int*)malloc(nbBits*sizeof(int));
291         presBin= tbBin(nbr);
292         printf("Votre nombre signe est : %d\n", c2EnDec(presBin,nbBits));
293     } else {
294         return;
295     }

```

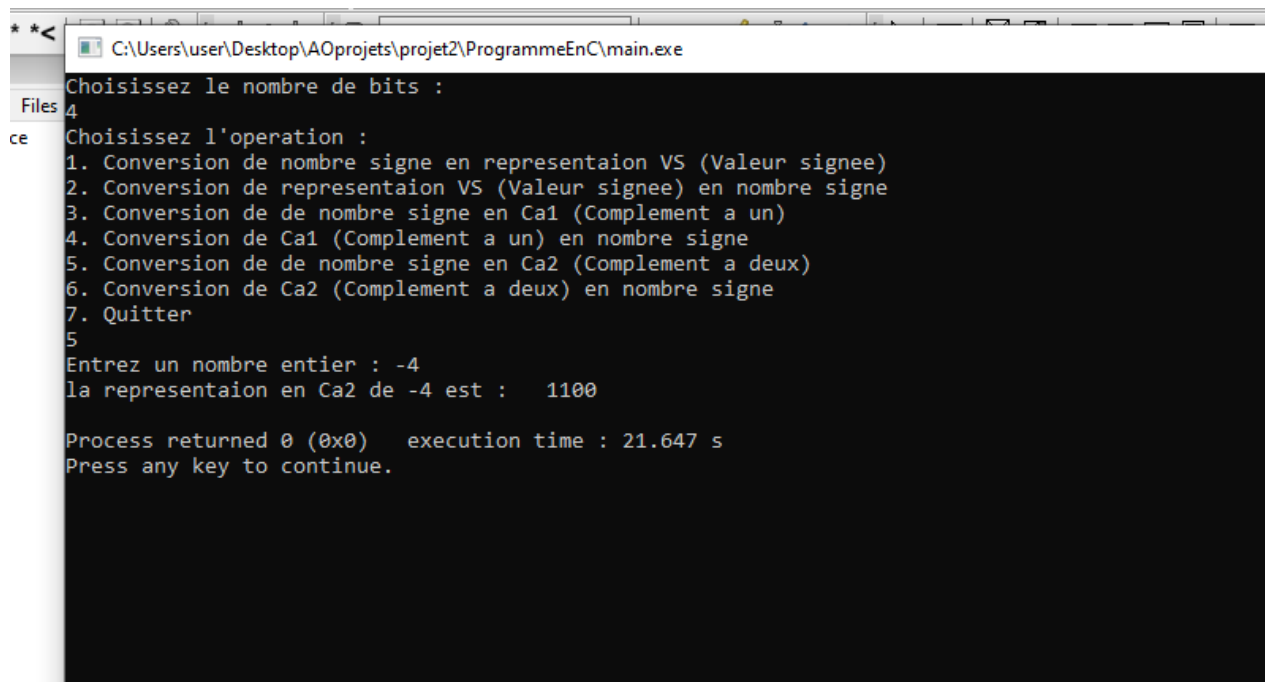
La fonction *convertirSigneMenu(int nbBits)* permet à l'utilisateur de choisir parmi différentes opérations de conversion impliquant des nombres signés et leurs représentations en Ca1 ou Ca2 et l'inverse. Les opérations comprennent la conversion entre les représentations et les nombres signés, avec des vérifications de débordement pour certaines opérations. L'utilisateur peut sélectionner une opération en entrant un choix, puis entrer les valeurs nécessaires pour effectuer la conversion.

15. Fonction main :

```
int main() {  
    int nbBits;  
    printf("Choisissez le nombre de bits : ");  
    scanf("%d", &nbBits);  
    convertirSigneMenu(nbBits);  
    return 0;  
}
```

Dans la fonction main on demande à l'utilisateur de spécifier le nombre de bits pour les conversions de nombres signés, puis appelle la fonction *convertirSigneMenu* pour effectuer les opérations de conversion en fonction du nombre de bits spécifié.

Voici un exemple d'exécution :



```
* * < C:\Users\user\Desktop\AOprojets\projet2\ProgrammeEnC\main.exe  
Choisissez le nombre de bits :  
4  
Choisissez l'operation :  
1. Conversion de nombre signe en representaion VS (Valeur signee)  
2. Conversion de representaion VS (Valeur signee) en nombre signe  
3. Conversion de de nombre signe en Ca1 (Complement a un)  
4. Conversion de Ca1 (Complement a un) en nombre signe  
5. Conversion de de nombre signe en Ca2 (Complement a deux)  
6. Conversion de Ca2 (Complement a deux) en nombre signe  
7. Quitter  
5  
Entrez un nombre entier : -4  
la representaion en Ca2 de -4 est : 1100  
  
Process returned 0 (0x0) execution time : 21.647 s  
Press any key to continue.
```