# Documentation en c : pour le développeur

**Sommaire :**



**Préambule :**

Un des principaux objectifs de notre code était de produire le plus clair et petit « main » possible, afin de rendre explicite notre code pour une personne venant à l'utiliser. Pour cela nous avons écrit dans notre « main » les principales fonctions qui seront en charge de la simulation de notre projet avec des noms très explicites. Un des principaux avantages de cette méthode est la détection presque automatique des éventuelles erreurs, en effet chaque fonction contient des potentiels messages d'erreur si la donnée venait à être illogique. Cela pourrait paraitre redondant mais le programme permet de détecter rapidement de quelle fonction provient l'erreur, cette méthode s'est avérée très efficace et nous a permis de gagner beaucoup de temps.

## I) Répartition du travail :

Pour réaliser notre projet nous avons d'abord bien discuté entre nous et essayé de bien comprendre le sujet. Nous avons ensuite élaboré un plan afin de se visualiser quelles fonctions seraient nécessaires dans notre code. De manière générale, nous avons procédé comme suit : d'abord nous avons défini les différentes fonctions principales dont nous allions avoir besoin puis pour chaque fonction principale on expose nos idées respectives, et on décide des éventuelles sous-fonctions dont on a besoin, ensuite nous nous sommes répartis les fonctions à coder selon nos convenances ( bien-sûr de manière équitable ), puis chacun a codé de son côté en effectuant des éventuelles améliorations, enfin nous nous somme regroupé et avons vérifié si tous les éléments fonctionnaient correctement .

## II) Explication des différentes fonctions (choix des fonctions et difficultés rencontrées)

Définissons la structure principale de notre programme nommé « nb ».

Cette structure contient deux paramètres : « valeur » qui est de type int et « instruction » qui est un tableau de « char » de quatre mots :

-case 0 : code opératoire.

-case 1 : le premier registre « rn ».

-case 2 : le deuxième registre « rd ».

-case 3 : le troisième registre ou valeur immédiate.

Cette structure nous sert principalement à créer notre tableau de mémoire, en effet nous avons conçu un tableau de pointeurs sur cette structure (de 65535 cases), chaque indice correspond à une adresse.

Chaque case du tableau contient :

- Une valeur qui correspond à une case mémoire de 1 octet.
- Un tableau de quatre mots, la case i (multiplie de 4 en comptant 0) contient les instructions du fichier qui correspond à la ligne i/4 du fichier assembleur.

Nous avons décidé de procéder ainsi afin de simplifier notre programme et d'éviter plusieurs traductions supplémentaires, notamment le passage du fichier hexadécimal aux instructions. En effet, au cours de la lecture de notre code par notre programme, on stocke nos instructions selon la structure définie précédemment, cela permet un gain de temps et de lignes de code.

Cette structure est également utilisée pour le tableau de registre sans utiliser le paramètre : "instruction". Notons que nous avons choisi un « int », pour vérifier lors d'une addition par exemple le dépassement du nombre sur 16 bits signé et ainsi pouvoir modifier notre registre « C ».

### 1) Fonctions de Vérification :

Nous avons rencontré des difficultés dans la vérification d'un fichier assembleur. En effet, vérifier la syntaxe de ce type de fichier s'est avérée laborieuse, car il a fallu tenir compte de plusieurs paramètres (présence d'espace à un endroit précis, vérification des étiquettes, des registres, etc.). Au début de l'écriture du code nous ignorions l'existence de la fonction « sscanf » et de ses fonctionnalités, ce qui a rallongé de plusieurs lignes de codes notre projet. Nous avons pris connaissance de cette fonction qu'après l'écriture de plusieurs lignes de code, nous avons tout de même essayé de l'intégrer à quelques fonctions, mais pas pour l'intégralité du programme car cela nous aurait retardé dans la suite de notre projet.

Malgré cette difficulté nous avons réussi à produire une fonction qui répond aux critères du cahier des charges.

Notre principale fonction de vérification se nomme : « verification_total_erreur_et_remplir_memoire ».

Elle permet également de remplir notre mémoire comme décrit plus haut et de vérifier tout type d'erreurs.

### 2) Fonction de traduction(en hexadécimal) :

Nous avions rencontré beaucoup de difficultés à écrire la fonction qui permet de traduire nos instructions en fichier hexadécimal. Au début, nous avions pensé naturellement à coder une fonction qui transforme notre instruction sur quatre octets en binaire en prenant chaque octet que l'on convertit en hexadécimal.

Cette fonction est très fastidieuse à écrire et il aurait fallu plusieurs lignes de codes. Après un temps de réflexions, nous avons pensé à la méthode suivante : on écrit le code opératoire décalé de 27 bits vers la gauche, on fait de même pour le registre de destination mais décalé de 22 bits vers la gauche, la source 1 décalée de 17 bits vers la gauche et le bit qui représente le cas d'un registre ou d'une valeur

immédiate est aussi décalé de 16 bits vers la gauche. On additionne ensuite le tout sans oublier la source 2 (représentée sur 16 bits qui correspondent aux 16 bits de poids faible de l'instruction).

Cette méthode assure que l'addition produit bien un nombre sur 32 bits qui représente l'instruction. Cette méthode est très courte et s'écrit en quelques lignes de codes, elle nous a aussi permis de mieux comprendre l'intérêt du décalage binaire et de son fonctionnement.

Notons que chaque octet a été traduit en décimale et placé dans la case mémoire qui lui correspond (l'octet de poids fort étant toujours placé à l'adresse i et l'octet de de poids faible à l'adresse i+3).

### 3) Fonctions d'instructions :

Une des difficultés rencontrées durant notre projet fût la manière de gérer les différents cas possibles lors d'une instruction (registre ou valeur immédiate, dépassement de l'intervalle de représentation…). Pour cela nous avons d'abord défini plusieurs sous fonctions qui permettront par la suite de réaliser des changements (mise à jour des bits N, C et Z et valeur du nombre dans le bon intervalle).

Nous avons décidé de subdiviser la principale fonction d'instruction en plusieurs sous fonctions qui permettront par la suite d'exécuter l'opération voulue. Ce choix permet de rendre plus claire le code, afin de limiter les risques d'erreurs. Ces fonctions font toutes appel à des fonctions de « vérification ». Par exemple si une donnée dépasse l'intervalle de représentation on le ramène à un bon intervalle ou bien si la donnée est stockée en mémoire on regarde si elle est valide.