TP 3 : Traduction de types de données avancés

Le but de cette séance est d'apprendre à traduire les types de données suivants : pointeurs, tableaux, chaînes de caractères et structures.

Les exercices sont organisés comme dans les TP précédents : dans le répertoire `tp3/` de votre dépôt, vous avez un fichier `exo.c` qui contient le programme principal, un fichier `fct_exo.s` à remplir, et une règle de génération dans le `Makefile` (`make exo`) pour générer l'exécutable. Dans tous les exercices, il convient de vérifier l'exécution pas à pas du programme avec le débogueur (`gdb`) et le simulateur (`qemu`).

Ex. 1 : Manipulation de chaînes de caractères

Cet exercice vise à traduire en langage d'assemblage RISCV des programmes C manipulant des chaînes de caractères.

Question 1 Dans le fichier `fct_taille_chaine.s`, donnez le contexte de la fonction `taille_chaine` et traduisez-la en langage d'assemblage RISCV. On rappelle qu'en C, les chaînes de caractères se terminent toujours par le caractère ASCII « \0 » dont la valeur est 0. Pour lire en mémoire un seul octet, on utilisera l'instruction `lbu` (Load Byte Unsigned) plutôt que l'instruction `lw` (Load Word).

Question 2 Exécutez votre code avec le simulateur comme dans les séances précédentes.

Au besoin, mettez au point le programme en utilisant GDB. Les formats d'affichage `/c` et `/s` des commandes `display` et `print` peuvent vous être utile pour afficher dans le débogueur respectivement un caractère ou une chaîne de caractères.

Question 3 En respectant le contexte fourni, traduisez en langage d'assemblage RISCV la fonction `inverse_chaine` du fichier `fct_inverse_chaine.s`.

Ex. 2 : Manipulation de tableaux

Le but de cet exercice est de travailler avec des tableaux d'entiers signés 32 bits. Le fichier de travail s'appelle `fct_tri_min.s`.

Question 1 Donnez le contexte de la fonction `tri_min` et traduisez-la en langage d'assemblage RISCV.

Ex. 3 : Manipulation de structures (agrégats)

Cet exercice permet d'appréhender la traduction de code C utilisant des types structurés (mot clef `struct`). La difficulté concerne les paramètres et/ou les valeurs de retour qui sont structurés, et non des pointeurs vers de tels types. En effet, les pointeurs sont des scalaires assimilables à des entiers (des adresses machines, en fait).

Question 1 Les fichiers `fct_struct.s` et `struct.c` contiennent des fonctions très simples utilisant un type structuré de taille inférieure ou égale à 64 bits.

Analysez ces fichiers pour répondre aux questions suivantes :
- Quelle convention impose l'ABI pour passer en paramètre des structures de cette taille ?
- Comment sont représentées les structures en mémoire ?
- Comment accède-t-on aux champs d'une structure à partir d'un pointeur vers cette structure ?

Pour aller plus loin...
Question 2 Que doit-on faire si le champ `entier` est remplacé par deux champs `p` et `q` de type `uint16_t` ? Même question s'il est remplacé par quatre champs `a, b, c, d` de type `uint8_t` ?

Question 3 Les fichiers `fct_double_rect.s` et `double_rect.c` contiennent des fonctions utilisant un type structuré de taille strictement supérieure à 64 bits.

Analysez ces fichiers pour répondre aux questions suivantes :
- Quelle convention utilise l'ABI pour récupérer une valeur de retour qui est une telle structure ?
- Quelle convention utilise l'ABI pour passer en paramètre une telle structure ?

Pour répondre à ces questions, vous devez regarder le code de `fct_double_rect.s`, mais aussi désassembler `double_rect`, l'exécutable résultant de la compilation, et regarder la manière dont les variables `rin` et `rout` sont allouées et utilisées. On fait ici ce qu'on appelle de la *rétro-ingénierie* de logiciel.

Question 4 Dans le fichier `fct_inverse_liste.s`, définissez le contexte de la fonction `inverse_liste`, puis implantez-la. On rappelle que `NULL` est codé par la valeur 0.

Pour aller plus loin...
Question 5 Implantez la fonction `decoupe_liste` du fichier `fct_decoupe_liste.s` en respectant le contexte imposé.

Pour aller plus loin...
Ex. 4 : Palindrômes

Cet exercice vous permettra à la fois de manipuler des chaînes de caractères et de faire des appels de fonctions. Le fichier de travail se nomme `fct_palin.s`.

Question 1 Donnez le contexte de la fonction `palin`, puis implantez cette fonction.

Attention : dans l'évaluation d'un « et » logique, si la partie gauche du « et » est fausse, la partie droite ne doit pas être évaluée [1].

1. Ceci permet par exemple lors d'un accès à un tableau de tester si l'indice respecte les bornes dans la partie gauche du « et » puis de faire un test sur la valeur accédée dans la partie droite du « et », le tout sans risquer une exception mémoire.