TD 4 Compilation

L3 INFO, Univ Lumière Lyon 2

2022 – 2023

Les exercices marqués avec (▶) sont notés, vous devrez les rendre pour évaluation. Utilisez les feuilles distribuées pour rendre votre copie. Les documents du cours sont autorisés. Les communications entre étudiants sont interdites.

▶ **Exercice 1** Représentez les nombres 31_{10}, 67_{10}, 111_{10} sous leur forme binaire. À partir de cette représentation binaire, vous en déduirez leur représentation hexadécimale. Attention, vos calculs pour l'obtention de la représentation binaire doivent être sur la feuille de réponse.

▶ **Exercice 2** Supposons que les valeurs suivantes soient stockées aux adresses de mémoire et aux registres indiqués (table à gauche). Compléter la table à droite avec les valeurs de chaque opérande.

Adresse	Valeur
0x120	0xAB
0x124	0xAB
0x128	0xF
0x12C	0x1

Registre	Valeur
%rax	0x3
%rcx	0x124
%rdx	0x2

Opérandes	Valeur
%rdx	
0x128	
$0x12C	
(%rcx,%rdx,2)	

TABLE 1 – État de la mémoire et registres (à gauche), valeurs à compléter (à droite).

▶ **Exercice 3** Considérez un fichier source nommé `sum.c` avec le contenu qui suit

```
1 long plus(long x, long y);
2
3 void sumstore(long x, long y, long *dest)
4 {
5   long t = plus(x, y) ;
6   *dest  = t ;
7 }
```

On veut lire la version assembleur de cette fonction.

1. Expliquez, en quelques mots, pourquoi la ligne du code ci-dessous produit une erreur

```
1 gcc -Og sum.c -o sum
```

2. Comment faut-il modifier la ligne pour pouvoir lire le code assembleur ?

Si vous le souhaitez, vous pouvez utiliser l'ordinateur pour reproduire l'erreur (vous pouvez aussi répondre sans besoin de coder).

Exercice 4 – attention cet exercice n'est pas noté Utiliser le code de l'exercice 3 comme base de travail.

1. Écrire la fonction `plus` qui renvoie la somme de deux arguments.
2. Compléter le code source afin d'obtenir un programme exécutable. Vérifiez que votre programme produit le résultat attendu si on initialise les variables avec de valeurs que vous choisirez. Enlevez de votre code tout affichage pour avoir un binaire le plus simple possible.
3. Obtenir le dump du fichier exécutable. Gardez-le dans un fichier.
4. Ouvrir l'exécutable avec `gdb`.
5. Désassemblez led code avec la commande `disassemble main`. Comparez l'affichage avec la section principale du fichier que vous avez créé dans 2.
6. Exécutez le code à l'intérieur de gdb en utilisant la commande `run`. Comprenez-vous ce qui se passe ? Pourquoi il n'y a aucun affichage ? Peut-on dire que l'ordinateur n'a rien fait ?
7. Utilisez la commande `breakpoint plus` pour placer un breakpoint au niveau de la fonction `main`.
8. Affichez l'état des registres avant l'exécution du code (commande `info registers`).
9. Placez maintenant un breakpoint au niveau de la fonction `plus`.
10. Examinez l'état de registres. Pouvez-vous identifier comment sont passés les arguments de la fonction ?
11. Vous pouvez examiner l'état d'une adresse mémoire avec la commande `examine *ADDRESSE`.

Pour vous aider à repérer les commandes les plus utilisées, voici un extrait de la documentation de `gdb`.

Command	Description
r	Start running program until a breakpoint or end of program
b fun	Set a breakpoint at the begining of function "fun"
b N	Set a breakpoint at line number N of source file currently executing
b file.c:N	Set a breakpoint at line number N of file "file.c"
d N	Remove breakpoint number N
info break	List all breakpoints
c	Continues/Resumes running the program until the next breakpoint or end of program
f	Runs until the current function is finished
s	Runs the next line of the program
s N	Runs the next N lines of program
n	Like s, but it does not step into functions
p var	Prints the current value of the variable "var"
set var=val	Assign "val" value to the variable "var"
bt	Prints a stack trace
q	Quit from gdb