SCR.2.1 TP 17 ⊥ :

Programmation en langage d'assemblage pour l'architecture ARMv8

https://developer.arm.com/documentation/ddi0602/2022-12/Base-Instructions

INPUT : Les nombres à tester sont placés dans la section `.data`. On peut utiliser `sed` et sa commande `s` pour en changer de valeurs. On retient leur adresse à l'aide d'étiquettes `int1`, `int2`, etc.

OUTPUT : Peut-on utiliser ce qu'on a fait au TP précédent lorsqu'on a affiché "Hello World !" ?

I. W vs X. Extension signée et non signée.

On va écrire `w-vs-x-and-extension.s` avec lequel on expérimente ce qui suit.

1. On veut tester ce qui arrive à la partie haute d'un registre `X` selon la forme du registre destination dans une instruction `ldr`.
   (a) Dans la section `.data` placer le nombre `0x11223344aabbccdd`. Retenir son adresse par l'étiquette `int`. Placer aussi le nombre `0x887766ee` et retenir son adresse par l'étiquette `int32`.
   (b) (*) Charger le 64-bit nombre dans le registre `x0`, puis (**) charger le 32-bit nombre dans le registre `w0`.
   (c) Dans `gdb`, placer un point d'arrêt juste avant l'exécution de (*). Consulter la valeur dans `x0` à chaque pas. Conclure.
2. On rappelle que si on a la représentation d'un nombre signé sur sur une longueur de codage donnée, alors sa représentation sur une plus grande longueur de codage s'obtient en dupliquant le bit de poids fort.
   (a) Dans `w-vs-x-and-extension.s`, juste avant de préparer les choses pour l'appel système `exit`, donc juste après (**), ajouter une instruction `sxtw` (Sign eXTend Word) qui, dans `x0`, étend le bit de signe de la partie `w0`.
   (b) On peut aussi étendre le bit de poids fort de l'octet de poids faible (`sxtb` : Sign eXTend Byte), le bit de poids fort du 16-bit mot de poids faible (`sxth` : Sign eXTend Halfword). Tester dans le même programme en opérant sur les mêmes registres. Voir ce qui passe dans `gdb`.
   (c) Ajouter dans le même programme ce qu'il faut pour tester ce que font les instructions `uxtb` et `uxth` pour Unsigned eXTend Byte et Unsigned eXTend Halfword respectivement.

II. Petits programmes en langage d'assemblage pour réaliser l'addition de deux nombres.

1. `add-int.s` qui fait une 64-bit addition.
2. `add-int32.s` qui fait une 32-bit addition.
3. `adds-int32.s` qui fait une 32-bit addition et qui met à jour les flags N,Z,C,V de l'état PSTATE du processeur.
   https://developer.arm.com/documentation/ddi0595/2021-06/AArch64-Registers/NZCV--Condition-Flags
4. Rappeler les conditions sous lesquelles peut se produire un débordement suite à une addition. Choisir alors en conséquence deux 32-bit entiers en entrée qui illustrent chacune de ces conditions. Le processeur a-t-il positionné le flag V (oVerflow) dans chaque cas ? Y a-t-il d'autre(s) flag(s) qui ont été positionnés par le processeur ?

La table des matières de la ressource suivante permet de naviguer assez facilement :
https://developer.arm.com/documentation/den0024/a