DIOP Abdoul Aziz
JAVAUD Corentin
ING3 CyberSécurité

Le 10 Décembre 2021

Rapport de Projet
Virtual Core

Intervenants :
Thomas TROUCHKINE - Guillaume BOUFFARD

Carte à Puce
Années 2021/2022

Sommaire

Implémentation du compilateur

Le compilateur a été implémenté en python. Ce choix s'explique par sa flexibilité et sa rapidité dans l'exécution de certaines instructions si on le compare à d'autres langages. Notre compilateur se charge de lire le code assembleur et de le traduire sous forme de code binaire en big endian qui sera décoder puis exécuter par le cœur. Chaque instruction sera encodée sous 32 bits avec une structure bien particulière.

Pour rentrer dans les détails, on lit chaque instruction (ligne par ligne) de notre code assembleur et on distingue les Branch instructions, des data processing instructions en utilisant la première lettre de l'instruction lue : B correspondant à la branche sinon c'est un data processing. Ainsi à partir de là, on les traite différemment.

Vérifier l'extension du fichier .s

Tout d'abord concernant les Branch instruction, on fait correspondre le valeur de la Branch condition Code à sa valeur en hexadécimal, cet valeur est ensuite mis au bit 28 d'une variable qui sera notre futur code. Pour l'offset, on vérifie préalablement son signe qu'on définit au bit 27 et du bit 0 au bit 27 on y place l'offset.

Concernant les data processing instructions, on distingue dans notre code 3 types d'instruction :

- les Compare (CMP) Instruction qui auront la particularité d'avoir qu'un opérande 1 (ope1) et un opérande 2 (ope2) ou une Immediate Value (IV).
- les MOV Instruction qui auront un registre de destination (dest), un ope2 ou une IV.
- et toutes les autre instructions restantes qui seront dans le même cas : un dest, une ope1 et une ope2/ IV

De ce fait, on encode notre as de cette façon en faisant des cas particuliers pour les Immediate Value. Dans le code, on traite les différents types de valeurs : les décimaux et les hexadécimaux.

Notre instruction alors créée est formatée en objet bytes grâce à la commande struct.pack avant d'être écrite dans le fichier binaire.

Implémentation du coeur virtuelle

Le cœur a été réalisé en C. Il est composé de 3 fonctions principales : fetch, decode et execute. Ces trois fonctions exécutées à la suite représentent un cycle d'instruction du cœur. Concernant l'architecture, nous utilisons 8 registres sous forme d'un tableau uint64_t.

Tout d'abord pour pouvoir exécuter les instructions de 32 bits il a fallu les récupérer à partir d'un fichier binaire. La fonction readfile lit le fichier binaire, récupère toutes les instructions et les convertit en BigEndian.

La fonction initRegisters lit également un fichier et elle initialise tous les registres. Nous avons rencontré quelques problèmes avec cette fonction et elle n'est pas parfaite. Afin que les registres soient bien initialisés il faut que la dernière ligne soit vide. Sinon le dernier registre n'a pas la bonne valeur.

La fonction fetch permet de savoir quelle est la prochaine instruction à exécuter. On vérifie d'abord si l'instruction est une instruction usuelle ou bien si nous sommes dans un cas BCC. On incrémente alors le program counter de 1 ou bien on récupère l'offset et son signe et on l'ajoute au pc.

On suit ensuite le cycle en passant dans la fonction decode sauf dans le cas BCC. Dans ce cas, on passe à l'instruction suivante. La fonction decode récupère les différentes parties de l'instruction.

On finit enfin par la fonction execute, cette fonction est un switch case de l'opcode de l'instruction avec deux cas différents selon la valeur du Flag de l'immediate value. Soit on fait les opérations avec la valeur du second operand ou bien avec l'"immediate value". On utilise les différentes opérations présentes en C et dans le cas d'un CMP on initialise BCCFlag en calculant chaques comparaisons possibles (BEQ, BNE, BLE, BGE, BL, BG).

Afin de lancer le coeur on utilise make all puis on saisit la commande suivante :
./prog <CODE> <STATE> (VERBOSE)
CODE correspond au binaire généré par le compilateur, STATE est l'état initial des registres
ATTENTION la dernière ligne du fichier doit être vide.
VERBOSE a 1 si on veut l'activer.

Programmes

Programme 1 : init_test

Le programme implémenté en pseudo assembleur permet de récupérer des valeurs sur 64bits et de les mettre dans des registres. Elle a pu être faite en utilisant des décalages vers la gauche (LSH). Ici l'utilisation de MOV n'a pas pu se faire car l'immédiate value est sur 8 bits, elle ne peut pas contenir les 64 bits à mettre dans le registre en une fois. Nous avons alors dû rajouter les valeurs 8 bits par 8 bits suivi d'un XOR pour les concaténer.

Programme 2 : add128_test

Pour ce programme, l'objectif est de faire une addition sur 128 bits alors que la taille n'est que de 64 bits. Pour ce faire nous avons séparé les 128 bits en deux parties et mis dans les registres r0, r1, r2 et r3 comme vu dans l'initial state. On a d'abord fait une addition sur r1 et r3 et un add avec retenue sur r0 et r2 dans le cas où la première addition en créerait (ce qui est le cas).

Le résultat se trouve dans le registre r4 et r5.

Programme 3 : 64 to 128 bits left shift

L'objectif ici était de faire un left shift d'une valeur sur 64 bits sachant que le résultat serait sur 128 bits. Pour se faire nous avons trouvé un moyen de faire un décalage de 1 bit sous forme d'addition pour ainsi récupérer le carry. Nous avons implémenté un programme qui grâce à une bcc instruction boucle 0xc (12) fois pour faire le add qui remplace le décalage et aussi ajouter les retenue dans un registre qui lui se chargera d'afficher l'excédent des 64 bits.

Le résultat se trouve dans le registre r3 et r1.

Réponses aux questions

1. Which parts of a 64 bits processor are 64 bits wits ?

Dans un processeur 64 bits, il s'agit des registres qui ont une taille de 64 bits.

2. Which instructions can potentially create a carry ?

Les instructions qui peuvent potentiellement créer un carry sont : ADD, ADC, SUB et SBC.

3. What is the purpose of the add carry (ADC) instruction ?

ADC instruction sert à additionner deux valeurs en y ajoutant la retenue qui était dans l'indicateur de retenue si il y en avait une. Il y a une valeur de retenue lorsque la largeur d'un nombre additionné ou soustrait à dépasser la largeur du processeur, 64 bits ici. Elle permet alors comme pour le programme 2 de faire des additions de plus de 64 bits dans un processeur n'en faisant que 64 bits.

4. What are the check to realize during branch instruction ?

Dans une branche instruction, il faut regarder les valeur des ope1 et ope2 du CMP dans l'instruction précédentes , les comparer avec le l'opération spécifique aux types de branche instruction, si c'est vrai on jump à partir de la valeur de l'offset.

5. It is possible to pipeline the virtual core

Malheureusement, on n'a pas pu trouver la réponse à cette question malgré nos recherches. Mais nous aimerions quand même la connaître si vous voulez bien nous la communiquer.