INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Université de Toulouse

RAPPORT
PROJET SYSTÈME INFORMATIQUE

UF Architecture logicielle et matérielle des systèmes informatique

Le Minh TRAN, Trong Hieu TRAN
Groupe 4IR - A1, INSA Toulouse

06 Juin 2019

SOMMAIRE

I. Introduction

Ce rapport contient le travail de TRAN Trong Hieu et TRAN Le Minh sur le projet Système Informatique, avec la présentation de la composition et du fonctionnement de deux parties : compilateur C et processeur VHDL.

II. Partie compilateur C

1. Composantes implémentées

a. Analyseur lexical et syntaxique

Conformément à la demande du cahier de charge, notre compilateur est capable de parser au niveau lexical des nombres entiers, des mots non commencer par un numéro ou un tiret bas *"_"*, les mots-clés *"if", "else", "while", "return","int","const","main",* les opérateurs de comparaisons *"==", ">=", "<=", ">", "<",* l'opérateur d'affectation *"="*, les opérateurs arithmétiques *"=","-","*","/",* les parenthèses *" ( ) "*, les accolades *" { } "*, le virgule *","* et point-virgule *";"*.

Au niveau syntaxique, le parseur peut parser les fonctions, l'entête d'un programme main, le corps du programme qui peut contenir les déclarations, des expressions d'affectations et arithmétiques, des expressions conditionnelles, des blocs IF-ELSE et WHILE et des pointeurs.

b. Table de symboles

La table de symboles sert à gérer les symboles trouver dans le corps du code, ainsi que leurs adresses et leurs profondeurs. A cette fin, le fichier TdS.h définit une structure Type_Symbole contenant 4 attributs : type du symbole, nom du symbole, profondeur, adresse, le fichier TdS.c définit un tableau interne de 100 Type_Symbole.

Pour manipuler la table de symboles, nous avons fourni les fonctions suivantes : **incr_profondeur()** et **decr_profondeur()** pour modifier la profondeur du symbole, **push(char * type, char * nom)** pour ajouter un symbole dans la table, **pop()** pour enlever la dernière ligne, **afficher_table()** pour afficher la table de symboles, **get_adr(char * nom)** pour prendre l'adresse d'un symbole grâce à son nom**, get_lastline_adr()** pour prendre l'adresse de la dernière ligne de la table.

c. Table d'instructions d'assembleur

Il y a un interpréteur qui sert à générer des instructions d'assembleur à partir du corps de code et les interpréter. A cette fin, le fichier interpreteur.h définit une structure ligneinter contenant 4 attributs : l'opérateur et les 3 registres utilisées. Le fichier interpreteur.c contient une table de 1000 ligneinter, un tableau de 1024 cases représentant la mémoire, un tableau de 16 cases représentant les registres.

Pour manipuler l'interpréteur, nous avons développé des fonctions suivantes: **ajout_ligneinter(char * op, int rA, int rB, int rC)** pour ajouter une instruction d'assembleur

dans la table, **afficher_tabinter()** pour afficher les instructions non interprétées, **interpreter()** pour réaliser des instructions d'assembleur, **update_rA(int index, int new_rA)** pour mettre à jour le registre rA de l'instruction qui se trouve à index, **from_registre(int r)** pour retourner la valeur contient dans le registre r, **current_index()** pour prendre l'indice actuelle de la table d'instruction.

2. Fonctionnalités détaillées des opérations

a. Opérations arithmétiques :

Pour effectuer une opération arithmétique, pendant le parsing, si on rencontre un nombre, on pousse une variable temporaire dans la table de symbole, puis générer des instructions d'assembleur pour stocker ce nombre dans l'adresse de la variable temporaire. L'idée de ces actions sont de stocker les deux membres d'une opération arithmétique dans des variables temporaires. Pendant le parsing des opérateurs arithmétiques, on fait deux instructions pour prendre les deux valeurs stockées dans les variables temporaires et les mettre dans les registres disponibles, à côté d' enlever des symboles temporaires, et puis effectuer le calcul entre ces deux registres et stocker le résultat dans l'un des deux. Il ne reste qu'à sauvegarder cette valeur trouvée dans l'adresse d'une nouvelle variable temporaire pour les affectations éventuelles ou les calculs suivants si l'opération contient plus de deux membres.

b. IF - ELSE

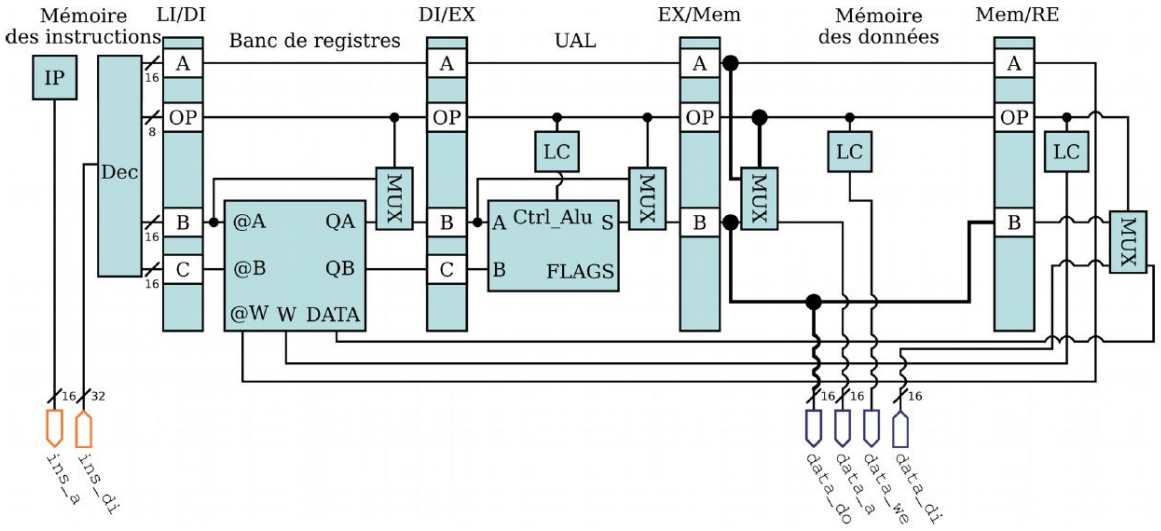
Pendant le parsing d'un bloc IF-ELSE, tout d'abord on effectue des opérations conditionnelles puis stocker le résultat dans le registre destiné à cette fin, dans notre cas c'est R10. Ce registre va contenir la valeur utilisée par l'instruction de saut conditionnel. Avant de parser le corps du IF, on sauvegarde la ligne de l'instruction de saut dans un drapeau, puis après avoir passé le corps du IF, on sait maintenant combien d'instructions il y a dedans, on peut donc mettre à jour la cible du saut conditionnel. Ainsi, si la condition du if est satisfait, on continue à effectuer les instructions suivantes, sinon on saute directement aux instructions qui ne se trouvent pas dans le corps du IF.

c. WHILE

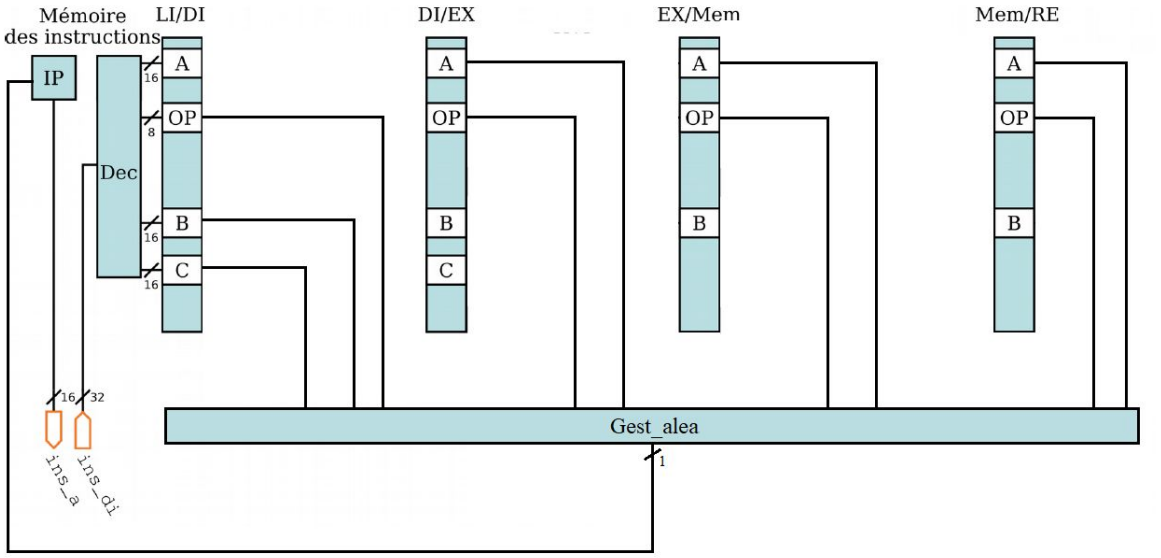
Pendant le parsing d'un bloc WHILE, avant d'effectuer des opérations conditionnelles, on sauvegarde la ligne de l'instruction de l'opération conditionnelle dans un premier drapeau, l'idée c'est de faire une instruction de saut à l'opération conditionnelle pour pouvoir toujours vérifier la condition du boucle. Le reste est pareil au fonctionnement du IF, sauf qu'il y aura un deuxième drapeau qui stocke la ligne d'une deuxième instruction de saut qui permet de, si la condition du boucle est satisfaite, on doit effectuer le contenu du boucle, sinon on sort de la boucle en sautant jusqu'aux instructions qui ne se trouvent pas dans ce contenu.

III. Partie processeur VHDL

1. Composition du processeur



Conformément au cahier de charge, nous avons implémenté un processeur composé d'une mémoire des instructions, un décodeur, un UAL, un banc de registres et banc de mémoire de données, avec différents niveaux de pipelines et des multiplexeurs pour rediriger les signaux en fonction des opérateurs. Nous avons aussi conçu une unité de gestion d'aléa présenté dans la figure ci-dessous. Actuellement notre processeur est capable d'effectuer des 4 opérations arithmétiques, la copie, l'affectation, le chargement et la sauvegarde.



2. Fonctionnalités des composantes en fonction de différentes opérations

Le processeur prend automatiquement les instructions depuis la mémoire des instructions en manipulant les "indices" du tableau : tous les 5 fronts montants de l'horloge, l'indice sera incrémenté d'un et la prochaine instruction est récupérée. L'instruction entre dans un décodeur pour extraire les opérateurs et les 3 registres utilisés avant d'entrer dans le premier niveau de pipeline - LI/DI. Sera propagé au prochain niveau - DI/EX la valeur de

B si c'est l'affectation ou le chargement, ou le contenu du registre B sinon. Avant le niveau EX/Mem, soit la valeur de B est propagée si c'est une affectation, un chargement ou une sauvegarde, soit la sortie d'UAL sinon. Avant le niveau Mem/Re, il y a un multiplexeur qui redirige la valeur de A dans l'entrée du banc de mémoire si c'est une sauvegarde, et la valeur de B sinon. Au dernier niveau, Mem/Re, l'entrée DATA du banc de registre va recevoir la valeur de la sortie du banc de mémoire si c'est un chargement ou une copie, et la valeur de B sinon.

IV. Conclusion

Nous avons accompli le projet Système Informatique comportant deux parties : le compilateur et le processeur, avec les exigences de fonctionnalité disponible satisfaite, le système global qui fonctionne correctement et qui donne des résultats attendus. Pour la présentation détaillée des résultats de l'implémentation de deux parties, veuillez consulter notre vidéo réalisé en complément avec ce rapport. Afin de voir plus précisément nos travaux, veuillez consulter le dépôt Github sur ce projet : https://github.com/kuro10/Projet_System_Info

On peut aussi réfléchir aux pistes d'amélioration pour la qualité de notre produit. Concernant le compilateur, nous avons encore les pointeurs et les fonctions qui ne sont pas implémentées, et nous pouvons aussi affiner le fonctionnement du bloc WHILE. Pour le processeur, nous avons réalisé une unité de gestion d'aléa qui est capable de détecter correctement des aléas, il nous reste donc d'implémenter le traitement du processeur face aux aléas, en plus des autres opérations telles que les comparaisons et le saut.

- Fin -