Bonjour à tous, aujourd'hui nous allons vous présenter notre soutenance du projet P-ARM, réalisé ce semestre.

Les objectifs de ce projet sont les suivants : Nous devons modéliser un modèle simplifié de processeur ARM cortex m0 sur le logiciel logisim évolution, le but à la fin étant de rendre ce processeur fonctionnel en lui faisant exécuter du code C compilé en assembleur.

Sommaire

Nous vous présenterons ainsi dans un premier temps le processeur, ainsi que ses différents composants. Nous énumérerons les tâches effectuées pour mener ce projet à bien. Ensuite, nous ferons une première démonstration testant notre processeur, ainsi que le programme assembleur que l'on aura réalisé. Nous parlerons de la couverture globale des tests du processeur. Puis nous listerons quelques points forts de notre processeur. Nous réaliserons finalement une seconde démonstration visant à exécuter un code C écrit par nos soins sur le processeur.

I – Présentation des composants

Notre processeur est constitué de 3 composants principaux :

1) L'ALU

Il s'agit de l'endroit dans le processeur chargé d'effectuer les calculs. Pour ce faire, il reçoit en entrée deux opérandes de 32 bits, ainsi que le calcul à effectuer. Il envoie en sortie le résultat du calcul, ainsi que des flags. Ces flags représentent une série d'états à la suite d'un calcul : un résultat négatif, un résultat nul, une retenue ou encore un débordement et servent dans les branchements conditionnels. Les deux opérandes en entrée sont envoyées par notre 2e composant

2) Le Banc de registres.

Le banc de registres sert de mémoire interne au processeur. Comme son nom l'indique, il est constitué de plusieurs registres. Dans notre projet, on en trouve 8 allant de R0 à R7 réservés à un usage général.

3) Le contrôleur

Le contrôleur joue le rôle de chef d'orchestre au sein du processeur. C'est lui qui est chargé du décodage des instructions reçues par le processeur. Ces instructions sont rédigées par le décodeur d'instruction vers le sous module correspondant.

Le data processing permet d'effectuer toutes les opérations arithmétiques entre deux registres et d'écrire le résultat dans un registre.
Le bloc shift, add, sub, mov permet d'effectuer des opérations de décalage, d'addition, de soustraction et de comparaison entre un registre et un immédiat. Le bloc a aussi la particularité de pouvoir stocker un immédiat dans un registre.

Le Stack Pointer est, dans notre cas, stocké dans un registre directement situé dans le contrôleur. Il stock l'adresse mémoire où démarre la pile. En général, on soustrait au Stack Pointer la quantité de mémoire dont on a besoin. Une fois l'utilisation terminée, on

l'incrémente pour revenir à la valeur initiale. Ces opérations sont effectuées au moyen du bloc SP Address qui permet d'ajouter ou retirer un immédiat à la valeur du Stack Pointer.
Le bloc Load/Store permet justement d'effectuer des transferts de données entre le registre et la mémoire. Ce bloc a la particularité d'être le seul dans notre projet à nécessité plus d'un cycle pour effectuer certaines opérations. En effet, la mémoire ne renvoie les données qu'au cycle suivant.

Le bloc Flags APSR conserve les flags en vue d'une potentielle utilisation au cycle prochain.
Le contrôleur gère aussi le Program Counter, qui détermine l'adresse de l'instruction suivante. Par défaut, il est incrémenté de 1 à chaque coup d'horloge.
Le conditionnal bloc permet justement d'interférer avec ce comportement. En effet, si la condition, codée dans l'instruction, est vérifiée par les flags qu'il reçoit du bloc Flags APSR, il effectue un certain saut dans l'exécution des instructions. L'immédiat qui lui est passé en paramètre permet de déduire ce décalage avec l'adresse de l'instruction actuelle.

II – Tâches effectuées

Pour réaliser ce projet, nous nous sommes répartis les différentes tâches de la manière suivante : la première semaine, Evan et Alban se sont occupés de réaliser l'ALU tandis que Théo s'est penché sur le fonctionnement du banc de registre. Le contrôleur a été réalisé lors des 2ème et 3ème semaine de travail, par l'ensemble du groupe.
La 4ème semaine, Théo s'est occupé du chemin de données tandis qu'Evan et Alban ont réalisé l'assembleur. Nous avons réglé les derniers problèmes de l'assembleur lors de la 5ème semaine, ce qui nous a permis de pouvoir exécuter du code c compilé dans le CPU.
Et enfin, lors de la dernière séance, nous avons ajouté différentes fonctionnalités pour approfondir le projet.

III – Démonstration : test CPU + Assembleur

tests\DataProcessing.txt

puis

Assembler -h

puis

SI3-PARM\code_asm\loop.bin

IV – Couverture globale des tests

V – Points forts de notre projet 1)

Comme dit précédemment, nous avons cherché à aller plus loin lors des derniers jours de travail pour approfondir le projet plus en détail, ce qui nous permet aujourd'hui de vous présenter ses points forts.

Tout d'abord, notre processeur est entièrement opérationnel. En effet, il est couvert par plus de 44 000 vecteurs tests que nous avons pu générer à l'aide d'un programme écrit par nos soins.

De plus, notre programme d'assembleur est fonctionnel, rendant possible l'exécution de code C sur notre processeur.

Nous avons aussi ajouté des entrées/sorties diverses, telles que des buzzers et des sliders, et nous les avons programmées.

VI – Démonstration d'un code C compilé passé au CPU

Compilation et exécution code joystick

VII – Point fort de notre projet 2)

Adressage indirect :
[introduire]

Tableau instruction : Donc voici notre tableau d'instruction, on peut voir qu'on a ajouté les instructions Load/Store avec registre et immédiat ainsi que Add SP-Relative. Par exemple, pour le Load, on est capable de stocker dans Rt la valeur située à l'adresse Rn + Imm. On peut aussi fournir un registre en tant qu'offset. Cette instruction est particulièrement utile pour itérer sur un tableau où Rn est l'adresse du premier élément du tableau et Rm le décalage depuis cette adresse pour aller à l'index voulu.

Pour faire cela, on a dû rediriger les bus A et B vers le contrôleur pour pouvoir calculer les adresses dans le Load/Store.

Montre pointers
Compile pointers
Execute pointers

Montrer audio.h
Montre rapidement buzzer_melody
Lancer vidéo (en disant que vitesse de sim trop faible sur pc portable pour avoir un rendu correct)