PROJET PROG 5

Notre projet se décompose en 5 fichiers, nous n'avons pas changé distinctement la structure du code fourni.

-registers.c et memory.c

Nous n'avons pas changé la structure de ces 2 fichiers, ils implémentent les interfaces de dialogue avec la mémoire simulée et les registres simulés.

-arm_instruction.c

arm_instruction est le cœur de notre code, ce fichier implémente la lecture des instructions chargées en mémoire et effectue un premier tri pour garantir la bonne exécution de notre programme arm.

arm_step()

Cette fonction appelle arm execute instruction et en fonction de son code de retour effectue un appelle au gestionnaire d'interruption (file arm_exeption.c)

arm_execute_instruction()

Une fois appelée, elle déclenche la procédure de lecture de la prochaine instruction (arm_fetch), vérifie que celle ci c'est bien déroulé et test ensuite si la condition d'exécution de l'instruction est bien valide via la fonction (check_flags). si toutes les conditions sont bonnes, elle appelle (switch_type)

switch_type()

Switch_type est la première fonction de décodage de l'instruction, celle-ci récupère les bits 25 à 27 de notre instruction et redirige vers le fichier correspondant pour la bonne exécution de l'instruction. (data processing, branchement, interface mémoire …)

check_flags()

-arm_data_processing.c

Il y a 2 type d'instructions que arm_instruction renvoie dans ce fichier,
-les instruction de data processing avec valeurs immédiates (arm_data_processing_immediate_msr)
-les instruction de data processing entre registres (arm_data_processing_shift)

arm_data_processing_shift()

Cette fonction récupère le 1er opérande (rn) et appelle la fonction (arm_shifter_op_data) pour le calcul du 2eme opérande (shifter_operand), puis apelle la fonction d'execution (arm_data_processing_operation)

arm_data_processing_immediate_msr()

Tout comme la fonction (arm_data_processing_shift) cette fonction récupère le 1er opérande mais aussi le 2nd, ici (immed_8) puis appelle la fonction d'exécution des instructions, (arm_data_processing_operation)

arm_shifter_op_data()

(arm_shifter_op_data) a pour tâche de calculer le 2nd opérande des instruction de data processing dans le cas d'une instruction de shift, elle change la valeur d'un pointeur index, avec la valeur de ce 2nd opérande.

arm_data_processing_operation()

Cette fonction contient l'implémentation de l'exécution des instruction de data processing, après récupération des informations nécessaires.

Dans certains cas ou l'on doit mettre à jour les flags, cette fonction appelle (update_flags)

update_flags()

Une simple fonctions de mise a jour des flags Z,N,C,V en fonction de l'instruction qui vient d'être exécuté

-arm_load_store.c

Deux fonctions principales peuvent être appelées depuis switch_type() de arm_instruction.c:

- *arm_load_store()*
- *arm_load_store_multiple()*

arm_load_store()

Cette fonction permet de traiter les toutes les instructions arm d'accès mémoire sauf LDM(1) et STM(1), les instructions d'accès mémoire multiples.

arm_get_address_word_byte()

Appelée par arm_load_store(), cette fonction permet de récupérer l'adresse à laquelle on stocke ou récupère une donnée de 32 ou 8 bits.

arm_ldr()

arm_str()

Les deux fonctions ci-dessus appelées par arm_load_store() permettent d'effectuer le stockage (str) ou la récupération (ldr) d'une valeur de 32 bits.

arm_ldrb()

arm_strb()

Les deux fonctions ci-dessus fonctionnent comme arm_ldr() et arm_str() mais pour une valeur de 8 bits.

arm_get_address_half()

Appelée par arm_load_store(), cette fonction permet de récupérer l'adresse à laquelle on stocke ou récupère une donnée de 16 bits.

arm_ldrh()

arm_strh()

Les deux fonctions ci-dessus fonctionnent comme arm_ldr(), arm_ldrb(), arm_str() et arm_strb() mais pour une valeur de 16 bits.

arm_get_index()

Cette fonction est appelée dans arm_get_address_word_byte() permet de récupérer l'index qui correspond au deuxième opérande auquel on effectue l'opération de décalage.

arm_load_store_multiple()

Cette fonction traite le cas des instructions d'accès mémoire multiples : LDM(1) et STM(1)

arm_get_start_end_address()

Cette fonction permet de récupérer l'adresse de départ et de fin qui serviront à effectuer les opérations LDM(1) et STM(1)

count_nb_set()

Cette fonction est appelée par arm_get_start_end_address() et permet de compter le nombre de bits à 1 dans un nombre passé en argument

arm_ldm()

arm_stm()

Les deux fonctions ci-dessus appelées par arm_load_store_multiple() permettent d'effectuer les instructions d'accès mémoire multiples LDM(1) et STM(1).

-arm_branch_other.c

arm_branch()

(arm_branch) est appelé suite à la détection d'une instruction de branchement, elle change la valeur de pc (r15) pour exécuter ce branchement.

arm_miscellaneous()

En armv5, la seule instruction miscellaneous est CLZ. Cette instruction renvoie le nombre de zéros binaires précédant le premier bit de valeur 1.

-arm_exeption.c

arm_exception()

Chaque fonction renvoie un code erreur particulier selon les macro définies dans arm_constant.h, la fonction arm_exception les récupère et fait le traitement adapté.