I. Tableau de bord

Jour	Action(s) faite(s)	Contributeur(s)
Jeudi 17/12 **Après la présentation**	Découvertes du sujet, et préparation des machines choix des conventions d'écriture (pour la documentation).	Vincent, Damien, Adrien, Yann, Nathan, Léandre
Vendredi 18/12	Début de la programmation des fichiers registres et memory plus planification du travail pour après les vacances.	Vincent, Damien, Adrien, Yann
Vacances	Avancement sur la mémoire	Vincent
Lundi 4/01	Il nous met au courant et chacun se familiarise avec ce qu'il a fait. Comprendre ce qui est fait dans les fichiers fournis. Et comprendre ce qu'il faut faire pour les différentes instructions.	Vincent, Damien, Adrien, Yann, Zakarya, Léandre, Nathan
Mardi 5/01	*arm_instruction* fait dans son intégralité, et début de *arm_data_processing*. *Arm_Load_et_store* commencé.	Vincent, Damien, Adrien, Yann, Léandre, Nathan
Mercredi 6/01	Branchement, reconstruction de arm instruction.	Vincent, Damien, Adrien, Yann, Léandre
Jeudi 7/01 **Audit de code 10h30**	Optimisation des fonctions dans le fichier *arm_load_store* et modification de *arm_instrcution*	Vincent, Damien, Adrien, Yann, Nathan, Léandre
Vendredi 8/01	Suite à l'audit de code : Optimisation des fonctions de mémoire et de registre ainsi que la factorisation de fonction dans *memory* et *registers*	Vincent, Damien
Lundi 11/01	Rectification de certain bug (entre autre *arm_instruction* , *arm_data_processing* et tentative de passage des différents tests	Vincent, Damien, (Adrien -> couvre-feu)
Mardi 12/01	Création des différents fichiers d'exemple, de test. Exécution du programme avec les différents tests et rectification des bug détecter (accès mémoire pour les bytes entre autres). Ajout du multi load/store (programmer par Adrien)	Vincent, Damien, Adrien, Yann
Mercredi 13/01	Correction des commentaires, dernier test et vérifications, début de la création des fichiers à rendre (compte rendu). Détail de ce que test et font les exemples, derniers tests de mise au point. Mise en commun.	Matin : Vincent, Damien, Adrien Après-midi : Vincent, Yann, Léandre, Damien, Adrien

Jeudi 14/01 **Rendu 12h**	Préparation du diaporama de présentation, dernière vérification du code et du compte-rendu.	Vincent, Yann, Léandre, Damien, Adrien, Nathan
Vendredi 15/01 à 12h10	Présentation !	

II. Les différents test/exemple pour l'exécution

Pour conclure sur les résultats, il faut suivre l'exécution des exemples et regarder l'état des registres et l'état de la trace après chaque instructions.

Fichier :	**Description :**
Example_branch.s	Permet de vérifier les fonctions de branchement fonctionne. Ici, nous testons : **bl, b**. Ici, nous réalisons un court programme afin de vérifier si la sauvegarde des registres fonctionne, si les sauvegarde de **lr** fonctionne et si nous pouvons revenir dans le programme principal. Ici, nous testons aussi les boucles et les test (par les branchements simples avec comparaisons)
Example_load_store.s	Permet de vérifier les différentes opérations sur la mémoire pour des opérations simple mais aussi les opérations double dans l'initialisation des variables. Ici, nous testons : **ldr, str, ldrb** etc… Pour cela, nous écrivons en mémoire avec ces différentes fonctions puis, nous récupérons ces valeurs dans un registre. A l'aide de gdb, nous pouvons voir dans les registres si les valeurs récupérer sont bien celles désirer, sous la forme désirer. Ce qui nous permet de vérifier si la lecture ou l'écriture en mémoire se fait correctement.
Example_mrs.s	Permet de tester la fonction **mrs**. Pour cela, nous regardons les valeurs dans les registres grâce à gdb, et pouvons vérifier si la valeur est bien celle présente dans le **CPSR** ou **SPSR** selon le cas dans l'exemple (les deux sont tester).
Example_multiple_load_store.s	Permet de vérifier les différentes opérations sur la mémoire pour des opérations multiple. Ici, nous testons : **ldmia, stmia, ldmdb** etc… Pour cela, nous écrivons en mémoire avec ces différentes fonctions puis, nous récupérons ces valeurs dans un registre. A l'aide de gdb, nous pouvons voir dans les registres si les valeurs récupérer son bien celles désirer, sous la forme désirer. Ce qui nous permet de vérifier si la lecture ou l'écriture en mémoire se fait correctement pour une opération multiple.
Example_shift.s	Permet de vérifier les différentes opérations de shifting. Ici, nous testons : **LSL, ROR** etc… Pour cela, nous écrivons des valeurs en registre que nous shiftons. A l'aide de gdb nous vérifions si les valeurs après shifiting est correct ou non.
Example_add.s	Permet de vérifier si tous les calculs sur data fonctionne avec des valeur direct (tester avec les **mov**) puis, vérifier chaque fonction entre deux registres simples avec des opérations unique. Ici nous testons : **add, sub, and** etc… Dans ce test, nous testons aussi les tests simples tel que : **tst, cmp** etc… A l'aide de gdb nous pouvons vérifier les valeurs dans le registre de retour pour le les opérations et le registre **CPSR** pour les tests simples.

III. Exécuter le programme

1) Récupérer les fichiers **.c** et **.h** et les **.s** ainsi que le **make.am**(dans Examples) dans le git.
    a. Télécharger les documents
    b. Copier-coller les documents dans le répertoire de travaille
2) Récupérer les fichiers des configurations et les make (tous les fichiers sauf les **.h** et les **.c**) dans les fichier fournis :
    a. Télécharger les documents fournis
    b. Copier-coller les documents dans le répertoire de travaille
3) Compiler :
    a. Se placer dans le dossier de travaille avec un inviter de commande
    b. Exécuter la commande **./chmod +wrx**
    c. Exécuter la commande **./configure CFLAGS='-Wall -Werror -g'**
    d. Exécuter la commande **./make**
4) Exécuter (avec gdb) :
    a. Se placer dans le dossier de travaille avec un inviter de commande en double (noter (1) et (2) par la suite)
    b. Dans (1) Exécuter la commande **./arm_simulator** ***ou***, pour suivre la trace de l'exécution, exécuter la commande **./arm_simulator --trace-registers --trace-memory** (pour faire du débogage)
    c. Dans (2) Exécuter la commande **./arm-none-eabi-gdb**
    d. Dans (2) Exécuter la commande **file Example/<name_example>** (pour le nom des différents exemples, se référer à la page précédente)
    e. Dans (2) Exécuter la commande **target remote localhost :<port>** (pour le port, veuillez vous référencer à la valeur du port pour gdb dans la fenêtre (1))
    f. Dans (2) Exécuter la commande **load**
5) Suivre l'exécution :
    a. Dans (2) Exécuter la commande **step** afin de pouvoir changer d'instruction
    b. (Optionnel) Dans (2) Exécuter la commande **info registers**, cette commande permet d'afficher les états des différents registres ainsi que leurs valeurs.

IV. Les différents bugs détecter et non résolu :

1) Bug lors de la fin du fichier, une erreur apparaît :

2) Il faut faire attention à la gestion de la mémoire, l'alignement est OBLIGATOIRE (il peut y avoir « des trous » (valeur vide, non utiliser) mais un **word** doit obligatoirement être sur un multiple de 32, un **half** sur un multiple de 8 et un **byte**, un multiple de 4.

V. Listes des fonctionnalités

Implémentées :

- Instruction de branchements : **B/BL,**
- Instruction de traitement des données : **AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC, TST, TEQ, CMP, CMN, ORR, MOV, BIC, MVN**
- Instruction d'accès à la mémoire : **LDR, LDRB, LDRH, STR, STRB, STRH, LDM, STM LDRSH, LDRSB ?**
- Instruction diverses : **MRS**
- Opération de décalage : **LSL, LSR, ASR, ROR, RRX**

Manquantes :

- La prise en comptes des exceptions
- La gestion des coprocesseurs

VI. Structures du code développé

Nom du fichier	Nom de la fonction	Utilité de la fonction
arm_instruction.c	arm_instruction()	Fonction de pivot sur les codes opérateurs des instructions. Fait appel à cond_fct().
	cond_fct()	Vérifie si l'instruction doit s'exécuter (en vérifiant le opcode dans les 4 premiers bit des instructions).
arm_branch_others.c	arm_branch()	Traite les fonctions de branchement simple **B/BL.**
	arm_miscellaneous()	Traite les fonctions diverses (dans notre implémentations actuel, uniquement l'instruction **MRS.**
arm_data_processing.c	arm_data_processing_shift()	Effectue tous les shift composé et simple avec une ou deux opérations.
	arm_data_processing_immediate()	Effectue les calculs lorsque les data sont en valeur immédiate.
	opcode()	Calcul entre 2 valeurs, la valeur de sortie et/ou les flags.
	verif_zero(), different(), negatif(), v_flag_add(), v_flag_sub()	Série de fonctions qui mets à jours les différents flags en fonctions des situations.
arm_load_store.c	arm_load_store()	Traite les instructions de load et store « simple » (**LDR, STR, LDRH, STRH, LDRSB, LDRSH, LDRB, STRB)**
	arm_load_store_multiple()	Traite les instructions de load et store multiple **(LDM, STM).**
	Est_Sys_Ou_User()	Vérifie si le processeur est en mode user ou système.

	Execution_...()	Fait le/les load ou store correspondant pour le format et dans le mode attendu.
	write_load_reg_mem()	Fait des appels de Executions… dans des cas « généraux ».