
## I. La description de la structure des fonctions que nous avons implémenté :

### 1) arm_data_processing.c

dans ce fichier nous avons découpé les 14 instructions d'ARM en sous fonction ainsi que les différents type de shift .les flags de ces 14 fonctions sont implémentées selon les principes décrits dans les manuels ARM page 115 et pour les principes d'implémentation du corps de ces fonctions et le même que la description qu'on trouve à la page 117 du manuel qui nous donne les différentes pages ou sont décrits comment implémenter ces fonctions.

Elles sont en plusieurs catégories :

- ces fonctions prennent en argument une structure ou d'un objet représentant le cœur (core) de l'architecture ARM P, une première operands op_gauche, une deuxième operands shifter_operand, un indicateur ou un drapeau (flag) qui spécifie si les conditions doivent être mises à jour dans le registre d'état (CPSR) après l'exécution de l'opération set_cond, un numéro du registre de destination où le résultat de l'opération AND sera stocké n_r_res et un carry c.Elles mettent à jour les flags en fonction des résultats et renvoient 0 si l'opération s'est déroulée correctement.Elles sont:

  **and(p,op_gauche,shifter_operand,n_r_res,set_cond,c);**

  **adc(p,op_gauche,shifter_operand,n_r_res,set_cond,c);**

  **sbc(p,op_gauche,shifter_operand,n_r_res,set_cond,c);**

  **rsc(p,op_gauche,shifter_operand,n_r_res,set_cond,c);**

  **orr(p,op_gauche,shifter_operand,n_r_res,set_cond,c);**

  **eor(p,op_gauche,shifter_operand,n_r_res,set_cond,c);**

- a la différence des fonctions ci-haut ces derniers prennent les même arguments à la seule différence du carry sortant qui n'est pas passé en argument mais il est géré dans le corps du fonction.Elles mettent à jour les flags en fonction des résultats et renvoient 0 si l'opération s'est déroulée correctement. Elles sont:

  **sub(p,op_gauche,shifter_operand,n_r_res,set_cond);**

  **rsb(p,op_gauche,shifter_operand,n_r_res,set_cond);**

  **add(p,op_gauche,shifter_operand,n_r_res,set_cond);**

  **bic(p,op_gauche,shifter_operand,n_r_res,set_cond);**

- ces fonctions prennent en argument une structure ou d'un objet représentant le cœur (core) de l'architecture ARM P, une seule operands shifter_operand, un indicateur ou un drapeau (flag) qui spécifie si les conditions doivent être mises à jour dans le registre d'état (CPSR) après l'exécution de l'opération set_cond, un numéro du registre de destination où le résultat de l'opération AND sera stocké n_r_res et un carry c. Elles mettent à jour les flags en fonction des résultats et renvoient 0 si l'opération s'est déroulée correctement.Elles sont:

  **mov(p,shifter_operand,n_r_res,set_cond,c);**

  **mvn(p,shifter_operand,n_r_res,set_cond,c);**

- ces fonctions prennent en argument une structure ou d'un objet représentant le cœur (core) de l'architecture ARM P, une première operands op_gauche, une deuxième operands shifter_operand, un numéro du registre de destination où le résultat de l'opération AND sera stocké n_r_res et un carry c.Elles mettent à jour les flags en fonction des résultats et renvoient 0 si l'opération s'est déroulée correctement.Elles sont:

  **tst(p,op_gauche,shifter_operand,n_r_res,c);**

  **teq(p,op_gauche,shifter_operand,n_r_res,c);**

- ces fonctions prennent en argument une structure ou d'un objet représentant le cœur (core) de l'architecture ARM P, une première operands op_gauche, une deuxième operands shifter_operand et un numéro du registre de destination où le résultat de l'opération AND sera stocké n_r_res .Elles mettent à jour les flags en fonction des résultats et renvoient 0 si l'opération s'est déroulée correctement .Elles sont:

  **cmp(p,op_gauche,shifter_operand,n_r_res)**

  **cmn(p,op_gauche,shifter_operand,n_r_res);**

- int Arithmetic_shift_right_by_immediate(uint8_t shift_imm, uint32_t rm,uint32_t *shifter_operand)

  Cette fonction effectue un décalage arithmétique vers la droite sur l'opérande spécifié par shifter_operand. Le décalage est défini par la valeur immédiate shift_imm.

- int Arithmetic_shift_right_by_register(uint32_t *shifter_operand,uint32_t rm,uint32_t rs,arm_core p)

  Cette fonction effectue un décalage arithmétique vers la droite sur l'opérande spécifié par shifter_operand. Le décalage est effectué en utilisant la valeur du registre rs comme décalage. Les résultats sont modifiés en conséquence, et les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p)

- int Rotate_right_by_immediate(arm_core p,uint8_t shift_imm,uint32_t rm,uint32_t *shifter_operand)

  Cette fonction effectue une rotation vers la droite sur l'opérande spécifié par shifter_operand. La rotation est effectuée en utilisant la valeur immédiate shift_imm. Les résultats sont modifiés en conséquence, et les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p)

- int rotate_right_by_register(uint32_t rs,uint32_t rm,uint32_t *shifter_operand,arm_core p)

Cette fonction effectue une rotation vers la droite sur l'opérande spécifié par shifter_operand. La rotation est effectuée en utilisant la valeur du registre rs comme décalage. Les résultats sont modifiés en conséquence, et les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p).

- int rotate_right_with_extend(arm_core p,uint32_t rm,uint32_t *shifter_operand)

  Cette fonction effectue une rotation vers la droite sur l'opérande spécifié par shifter_operand, en utilisant une extension. Les résultats sont modifiés en conséquence, et les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p).

- int immedate(arm_core p , uint8_t immed_8 ,uint8_t rotate_imm , uint32_t *shifter_operand )

  Cette fonction est liée à un décalage immédiat, en utilisant les paramètres immed_8 et rotate_imm. Les résultats sont modifiés en conséquence, et les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p).

- int logical_shift_left_by_immediate(arm_core p , uint32_t shift_imm , uint32_t Rm , uint32_t *shifter_operand)

  Cette fonction effectue un décalage logique vers la gauche sur l'opérande spécifié par shifter_operand. Le décalage est défini par la valeur immédiate shift_imm. Les résultats sont modifiés en conséquence, et les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p).

- int logical_shift_left_by_register(arm_core p , uint32_t Rm ,uint32_t *shifter_operand ,uint32_t Rs_value)

  Cette fonction effectue un décalage logique vers la gauche sur l'opérande spécifié par shifter_operand. Le décalage est effectué en utilisant la valeur du registre Rs_value comme décalage. Les résultats sont modifiés en conséquence, et les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p)

- int logical_shift_right_by_immediate(arm_core p , uint32_t shift_imm , uint32_t *shifter_operand , uint32_t Rm)

  Cette fonction effectue un décalage logique vers la droite sur l'opérande spécifié par shifter_operand. Le décalage est défini par la valeur immédiate shift_imm. Les résultats sont modifiés en conséquence, et les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p).

- int logical_shift_right_by_register(arm_core p , uint32_t Rs , uint32_t Rm ,uint32_t *shifter_operand)

  Cette fonction effectue un décalage logique vers la droite sur l'opérande spécifié par shifter_operand. Le décalage est effectué en utilisant la valeur du registre Rm comme décalage. Les résultats sont modifiés en conséquence, et les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p)

- int carry(uint32_t ins, uint32_t *shifter_operand,arm_core p)

  La fonction Carry gère le drapeau de retenue (carry) en fonction de l'instruction spécifiée par ins. Les résultats sont modifiés en conséquence, et les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p).

- void nzc_shifftercarry_update (arm_core p, uint32_t result, int c)

  Met à jour les drapeaux N, Z, et C en fonction du résultat de l'opération spécifiée par result. Les modifications sont également appliquées à la structure représentant le cœur (core) ARM (arm_core p).

2) **arm_branch_other.c**
   - int arm_branch(arm_core p, uint32_t ins) : Gère les instructions de branchement (B, BL, BX, BLX).
   - int arm_miscellaneous(arm_core p, uint32_t ins) : Gère les instructions diverses, en particulier la lecture des registres CPSR/SPSR
3) **arm_load_store.c**
   - arm_load_store(arm_core p, uint32_t ins) : Cette fonction est la fonction principale pour les opérations de chargement et de stockage. Elle détermine le type d'instruction à partir du code opération et appelle la fonction appropriée pour traiter cette instruction.
   - determine_instruction_type(uint32_t ins) : Cette fonction détermine le type d'instruction (LDR, STR.) à partir du code opération.
   - is_scaled_register_offset(uint32_t ins), is_register_offset(uint32_t ins) et is_immediate_offset(uint32_t ins) : Ces fonctions vérifient le mode d'adressage de l'instruction, tel que le décalage immédiat, le décalage de registre ou le décalage de registre échelonné.
   - handle_memory_operation(arm_core p, uint32_t ins, uint32_t write_back) : Cette fonction effectue les opérations de chargement et de stockage en utilisant les différents modes d'adressage.
   - calculate_address(arm_core p, uint32_t Rn, uint32_t offset, uint32_t U) : Cette fonction calcule l'adresse effective à partir du registre de base, du décalage et de la direction d'incrémentation/décrémentation.
   - process_memory_access(arm_core p, uint32_t address, int load_store, int B, uint32_t Rd) : Cette fonction effectue l'accès à la mémoire en fonction de l'opération de chargement ou de stockage, de la taille du transfert et du registre destination.
   - handle_ldr_str(arm_core p, uint32_t ins) : Cette fonction gère les opérations de chargement (LDR) et de stockage (STR) en fonction du mode d'adressage.
   - handle_ldrh_strh(arm_core p, uint32_t ins) : Cette fonction gère les opérations de chargement (LDRH) et de stockage (STRH) en fonction du mode d'adressage.
   - initialize_common_variables, initialize_immediate_variables, initialize_register_variables, initialize_scaled_register_variables : Ces fonctions initialisent les variables communes utilisées dans les différentes fonctions de gestion des modes d'adressage
4) **arm_instruction.c**
   - **ConditionPassed (dans arm_constants.h) :**
     - ➢ Vérifie si la condition pour exécuter une instruction est satisfaite.
   - **arm_execute_instruction (dans le fichier source) :**
     - ➢ Fonction principale de traitement des instructions ARM.
     - ➢ Effectue le fetch de l'instruction, détermine la catégorie, vérifie les conditions, et appelle la fonction appropriée pour l'exécution.
   - **arm_step (dans le fichier source) :**
     - ➢ Fonction appelée pour exécuter une seule étape d'instruction.

Appelle arm_execute_instruction et gère les exceptions si nécessaire

## II. Liste des fonctionnalités implémentées

### 1) arm_data_processing.c

- Instructions de décalage et opérations logiques (ASR, LSL, LSR, ROR, RRX).
- Opérations arithmétiques (ADC, SBC, RSC, ADD, SUB, RSB).
- Opérations logiques (AND, EOR, ORR, BIC, MVN).
- Comparaisons et tests (TST, TEQ, CMP, CMN).
- Déplacement de données (MOV).
- Mise à jour des drapeaux (N, Z, C).
- Gestion des instructions de décalage par registre et immédiat

### 2) arm_branch_other.c

- Cette fonction gère  les instructions de branchement immédiat (B, BL) avec ou sans lien.
- et elle gère aussi les instructions de branchement enregistrement (BX, BLX) avec différentes conditions
- Lecture et écriture des registres CPSR et SPSR en fonction du bit R dans l'instruction

### 3) arm_load_store.c

- **Opérations de Chargement et de Stockage :**
  - Les opérations de chargement (LDR) et de stockage (STR) sont partiellement implémentées.
  - Les différents modes d'adressage (immédiat, registre, registre échelonné) sont pris en charge.
- **Opérations de Chargement et de Stockage Halfword (LDRH, STRH) :**
  - Certaines opérations de chargement et de stockage pour les halfwords sont implémentées (handle_ldrh_strh).
- **Opérations Load/Store Multiple :**
  - La fonction arm_load_store_multiple est partiellement implémentée pour les instructions STM et LDM

### 4) arm_instruction.c

- **Instructions de Traitement des Données :**
  - Opérations arithmétiques et logiques sur des données immédiates.
  - Opérations arithmétiques et logiques sur des données avec décalage.
- **Instructions de Chargement/Stockage :**
  - Instructions de chargement et de stockage immédiat.
  - Instructions de chargement et de stockage avec décalage.
  - Instructions de chargement et de stockage multiples.
- **Instructions de Branchement :**
  - Branchement conditionnel.
  - Branchement inconditionnel.
  - Branchement avec lien (BL).
- **Instructions Coprocesseur :**
  - Instructions de chargement/stockage du coprocesseur.
  - Autres instructions du coprocesseur et interruptions logicielles (SWI).
- **Instructions Diverses :**
  - Instructions diverses, telles que MRS (Move to Register from Status)

## III. Liste des fonctionnalités  manquantes

### 1) arm_data_processing.c

- rien de manquant

### 2) arm_branch_other.c

- ❖ Gestion des Interruptions et Exceptions :
  - ➢ Il n'y a pas de gestion apparente des interruptions ou des exceptions dans le code. Les architectures ARM implémentent généralement des mécanismes pour gérer ces événements.

3) **arm_load_store.c**
   - ❖ **Opérations Load/Store pour Coprocesseurs :**Les opérations de chargement et de stockage pour les coprocesseurs (arm_coprocessor_load_store) ne sont pas implémentées.
4) **arm_instruction.c**
   - ❖ rien de manquant

## IV. <u>Liste et description des tests effectués ;</u>

### 1) arm_data_processing

- ❖ Test generaux :
  - ➢ Vérifier le bon appel de chaque fonction lors de l'appelle à la fonction
  - ➢ Vérifier que la fonction carry retourne la bonne retenue
- ❖ **Test les fonctions de traitement de données:**
  - ➢ **Instruction arithmétique :**
    - ■ add/sub/rsb/adc/sbc/rsc:Tests avec des valeurs positives, négatives et des limites d'entiers,En surveillant attentivement la gestion des retenues, des dépassements, et en vérifiant les indicateurs de statut après chaque opération.
  - ➢ **Instruction logique :**
    - ■ and/eor/tst/teq/orr/bic:Les tests ont impliqué des vérifications avec une variété de valeurs d'entrée, y compris des valeurs maximales,minimales et aléatoires, permettant de garantir que ces instructions produisent les résultats attendus dans une large gamme de situations. Les combinaisons spécifiques de bits ont été testées pour vérifier que seuls les bits désirés sont activés ou désactivés, tandis que les opérandes avec des bits nuls ou tous les bits à 1 ont été utilisés pour confirmer les comportements attendus.
  - ➢ **Instruction de comparaison :**
    - ■ cmp/cmn:Des comparaisons directes entre des valeurs positives,négatives et zéro ont été effectuées à l'aide de l'instruction CMP. En utilisant des branches conditionnelles telles que BEQ, BNE, BGT,BLT, le comportement du simulateur a été évalué pour vérifier si les sauts conditionnels se produisaient conformément aux résultats attendus des comparaisons. Pour l'instruction CMN, des tests de complément à deux entre valeurs positives et négatives ont été réalisés, tout en surveillant attentivement la mise à jour des indicateurs de drapeaux.

### 2) arm_branch_other.c

- ❖ **Test généraux de branch:**
  - ➢ Branchement en avant
  - ➢ Branchement en arrière
  - ➢ Branchement vers une adresse spécifique
  - ➢ Branchement hors des limites
- ❖ **Test spécifique pour b:**
  - ➢ Branchement conditionnel(bne,be,bgt...)
  - ➢ Brancher avec un immediate
  - ➢ Brancher avec une étiquette
- ❖ **Test spécifique pour bl:**
  - ➢ Vérifiez que pc a changer avec l'adresse de l'instruction après le branch

- ➢ Brancher avec un immediate
- ➢ Brancher avec une étiquette
- ➢ Tester le retour de branch

❖ **Test spécifique pour bx :**

- ➢ Brancher avec un registre

❖ **<u>Test spécifique pour blx :</u>**

- ➢ Vérifiez que pc a changer avec l'adresse de l'instruction après le branch
- ➢ Brancher avec un registre
- ➢ Tester le retour de branch

**3) load_store**

- ❖ test par un registre et une valeur immédiate
- ❖ test avec deux registre
- ❖ test entre deux valeur immédiates

## V. Liste des éventuels bogues connus mais non résolus

**Nous n'avons pas de bugs**

## VI. Journal décrivant la progression du travail et la répartition des tâches au sein du groupe

| nom des participants | parties impliquées ou réalisées |
|---|---|
| **Haya Samodi Amin** | Data-processing (Codage / Correction des bugs), branch (Correction des bugs) et tests, mémoire |
| **Mohamed Amine El ouechrine** | coder et tester les instructions branch , Data-processing, registers |
| **Ismail Abdraman Abdeldjabar** | Data-processing,registre, descriptif des fonctions,listes des fonctionnalités implémenter et manquants,journal de progression |
| **messipsa bousaid** | memoire,arm_data_processing_functions,exceptions |
| **madhbouh yassin** | arm_load_store , load_store_addressing_operations , load_store_memory_operations , arm_instructions,registres , exception , automatisation des test avec gdb |
| **oueslati yosri** | arm_load_store_multiple, memoire, MSR, MRS |

[1]

[1] Merci pour votre lecture 👍