Guide de mise en oeuvre d'une première instruction

E2i5 – Polytech'Grenoble

Février – Juin 2023

Toutes les informations contenues dans ce guide font référence au cahier des charges (cdc) du projet. Récupérez et lisez ce document avant de continuer.

1 Les questions essentielles

Les questions suivantes doivent être répondues à chaque fois que vous souhaitez mettre en oeuvre une nouvelle instruction dans votre processeur. Consultez la méthode de conception (cdc, Section 4) et répondez soigneusement en utilisant comme exemple l'instruction `lui`.

Question 1.1. Avez-vous identifié l'instruction à implémenter et son format ? (registres impliqués, type d'immédiat, exemple en assembleur, ...) (cdc, Section 4.1)

Question 1.2. Quels sont les composants de la PO nécessaires pour mettre en oeuvre l'instruction ? (cdc, Section 4.2)

Question 1.3. Quels sont les signaux de la PO qui doivent être affectés pour mettre en oeuvre l'instruction ? (cdc, Section 4.2)

Question 1.4. Combien d'états sont nécessaires dans la PC pour mettre en oeuvre l'instruction ? (cdc, Section 4.3).
Attention : les accès à la mémoire prennent un cycle (voir signaux `mem_dataout` et `mem_datain`)

2 Mise en oeuvre de l'instruction `lui`

Toutes les modifications proposées dans cette section doivent être faites dans le fichier `CPU_PC.vhd`

2.1 Déclaration d'un nouvel état

— Identifiez le type énuméré `State_type` défini dans votre partie contrôle
— Modifiez ce type et ajoutez un nouvel état correspondant à l'instruction `lui`

```
type State_type is (
    S_Error,
    S_Init,
    S_Pre_Fetch,
    S_Fetch,
    S_Decode,
    -- Declaration du nouvel etat
    S_Lui
);
```

2.2 Modification de l'état `S_Decode` afin de décoder l'instruction

— Consultez l'encodage des instructions RISC-V (cdc, annexe F.3) et observez la colonne la plus à droite correspondant à l'opcode des instructions.

  — Quels sont les bits communs dans l'opcode des instructions ?
  — Quels sont les bits permettant de distinguer une instruction en format B d'une instruction en format I ?
  — Quels sont les bits permettant de distinguer l'instruction `addi` de l'instruction `andi` ?
  — Quels sont les bits permettant de distinguer l'instruction `add` de l'instruction `addi` ?

— Les questions précédentes vous mènent à une réflexion permettant de choisir une stratégie de décodage dans votre PC. Chaque binôme peut choisir une stratégie différente, mais essayez toujours de vérifier le plus petit nombre de bits possible.

Exemple : Décodage de l'instruction `lui`

Deux observations peuvent être faites lors de l'analyse concernant l'instruction `lui`

— Il est possible de distinguer les instructions en format U à l'aide des bits `4...2` du registre d'instruction (`IR` dans la PO).
⇒ **Ceci correspondra à la première étape de décodage**

— Il est possible de distinguer l'instruction `lui` de l'instruction `auipc` à l'aide des bits `6...5` du registre `IR`.
⇒ **Ceci correspondra à la deuxième étape de décodage**

Suite à ces observations, vous pouvez décoder votre instruction `lui` en modifiant l'état `S_Decode` de votre PC.

```
when S_Decode =>
    -- PC <- PC + 4
    cmd.TO_PC_Y_sel <= TO_PC_Y_cst_x04;
    cmd.PC_sel <= PC_from_pc;
    cmd.PC_we <= '1';
    -- prochain etat par defaut apres S_Decode
    state_d <= S_Error;

   -- Decodage effectif des instructions (calcul du vrai state_d)
    case status.IR(4 downto 2) is

        -- Instructions avec immediat de type U
        when "101" => -- lui ou auipc
            -- Identification entre lui et auipc
            case status.IR(6 downto 5) is
                when "01" => -- lui
                    state_d <= S_Lui;
                -- l'option par defaut (dans le cas ou les autres ne sont pas encore decodees)
                when others => null;
            end case;

        -- l'option par defaut (si les autres cas ne sont pas encore couverts)
        when others => null;
    end case;
```

L'idée générale dans l'état de décodage est de déterminer quel est l'état (`state_d`) vers lequel on veut aller au prochain cycle de la FSM. Ceci est déterminé toujours par certains bits de l'instruction stockée dans le registre `IR` de la PO.

2.3 Mise en oeuvre des nouveaux états dans la PC

Les réponses aux questions essentielles (Section 1) vous permmettent d'écrire votre code `vhdl` et d'affecter correctement les signaux de commande de la PO, utilisés ensuite pour contrôler l'exécution de l'instruction `lui` (cdc, Section 4.4).

```
when S_Lui =>
    -- rd <= ImmU + 0
    cmd.PC_x_sel <= PC_X_cst_x00;
    cmd.PC_y_sel <= PC_Y_immu;
    cmd.RF_we    <= '1';
    cmd.DATA_sel <= DATA_from_PC;
    -- lecture mem[PC] (de la prochaine instruction en memoire à l'adresse PC)
    cmd.ADDR_sel <= ADDR_from_PC;
    cmd.mem_ce   <= '1';
    cmd.mem_we   <= '0';
    -- next-state (apres lui)
    state_d <= S_Fetch;
```

Attention : Les signaux affectés dans l'état `S_Lui` remplaceront les valeurs par défaut initialisées au début de votre processus `FSM_comb`.

2.4 Mise en oeuvre du test en langage assembleur

Les tests correspondants à chaque instruction ajoutée dans votre PC doivent être placés dans le dossier `program/autotest/`
Le test de l'instruction `lui` est donné à titre d'exemple.

```
# TAG = lui
    .text

    lui x31, 0         #Test chargement d'une valeur nulle
    lui x31, 0xfffff   #Test chargement d'une valeur maximal sur 20 bits
    lui x31, 0x12345   #Test chargement d'une valeur quelconque

    # max_cycle 50
    # pout_start
    # 00000000
    # FFFFF000
    # 12345000
    # pout_end
```

Attention :

— L'écriture du résultat final de l'instruction est faite dans le registre `31` parce que c'est le registre envoyé sur les leds de la carte FPGA. En simulation, vous pouvez utiliser n'importe quel autre registre (`x1...x31`).

— Le `TAG` et les commentaires (`#`) à la fin du fichier correspondent au mécanisme d'autotest qui vous permettent de valider votre instruction (cdc, annexe D.3.2)

— Le mécanisme d'autotest vérifie uniquement les valeurs écrits dans le registre `31`

3 Validation de l'instruction

3.1 Validation par simulation

— Exécutez la commande permettant de démarrer une simulation (cdc, annexe D.1)
`$ make simulation TOP=PROC PROG=lui`

— Attendez les étapes d'analyse, elaboration et démarrage du xsim (voir Figure 1)

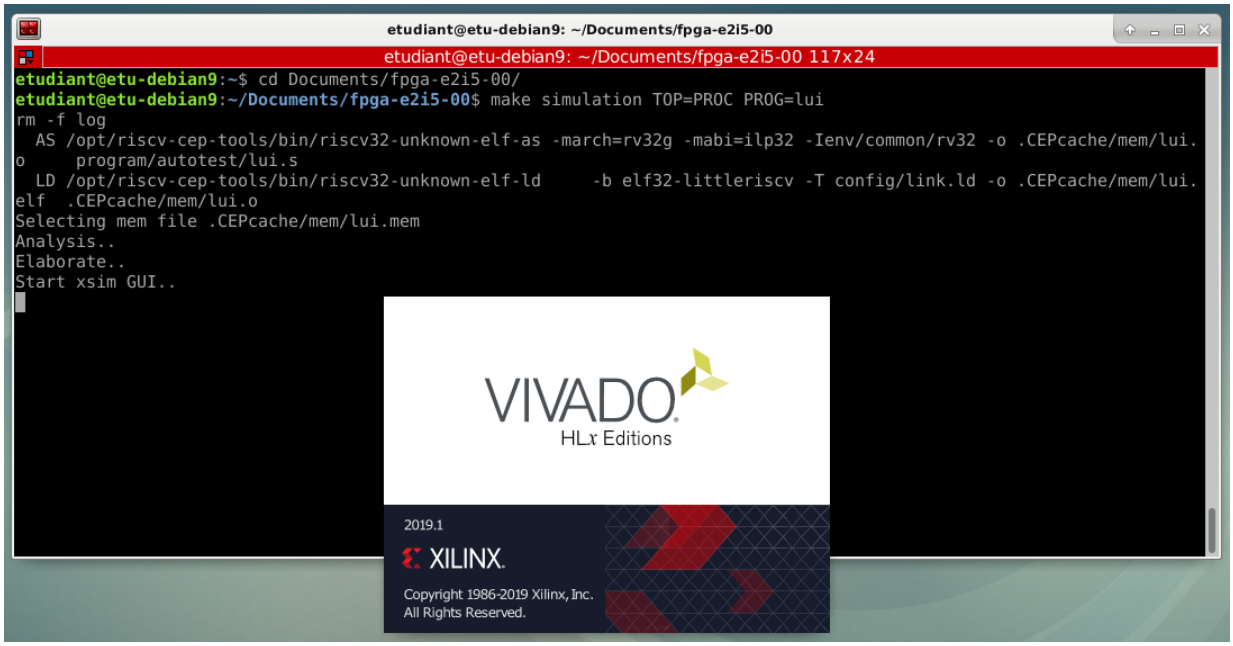

Figure 1 – Exécution de la simulation – test `lui.s`

— Validez les résultats de votre instruction (voir Figure 2). Observez les transitions de l'état `state_q` ainsi que la sortie `pout_q` (la même du RF_{31})

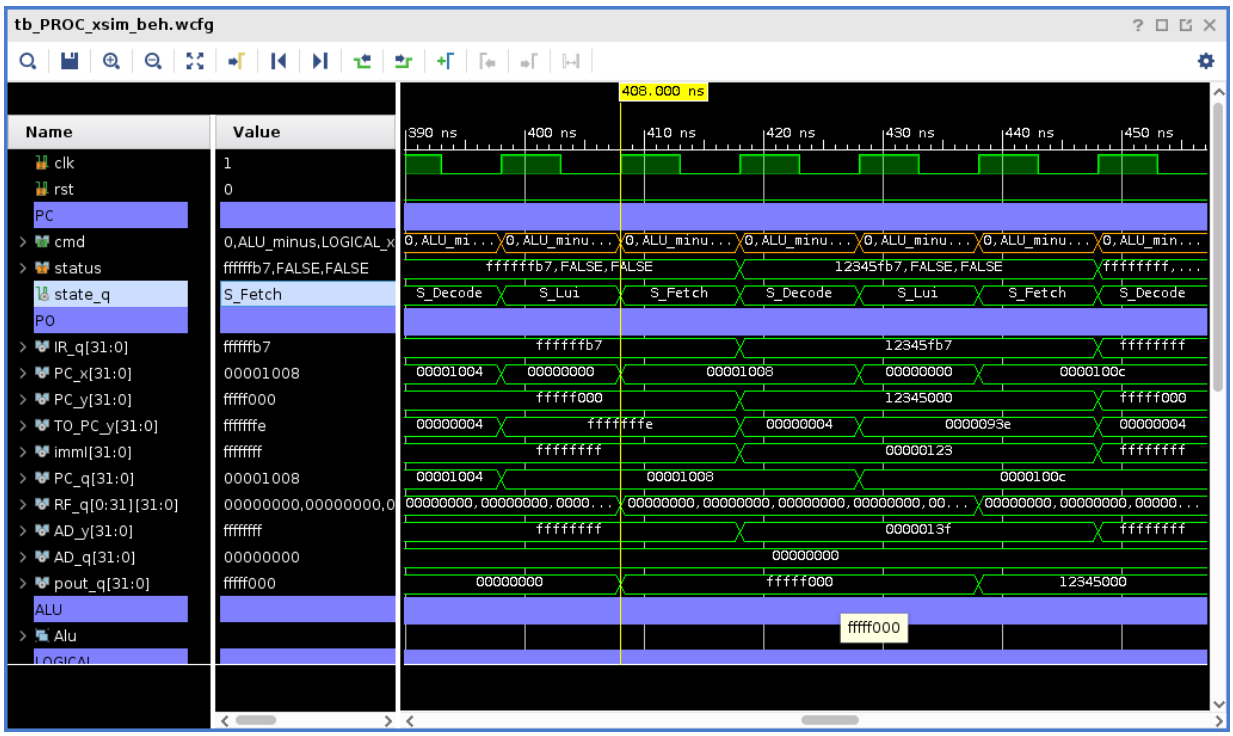

Figure 2 – Validation des résultats de l'instruction `lui` par simulation

3.2 Validation automatique par le mécanisme d'autotest

— Exécutez la commande permettant de démarrer l'autotest (voir Figure 3)
`$ make autotest`

— Validez le résultat de votre test (`PASSED`).

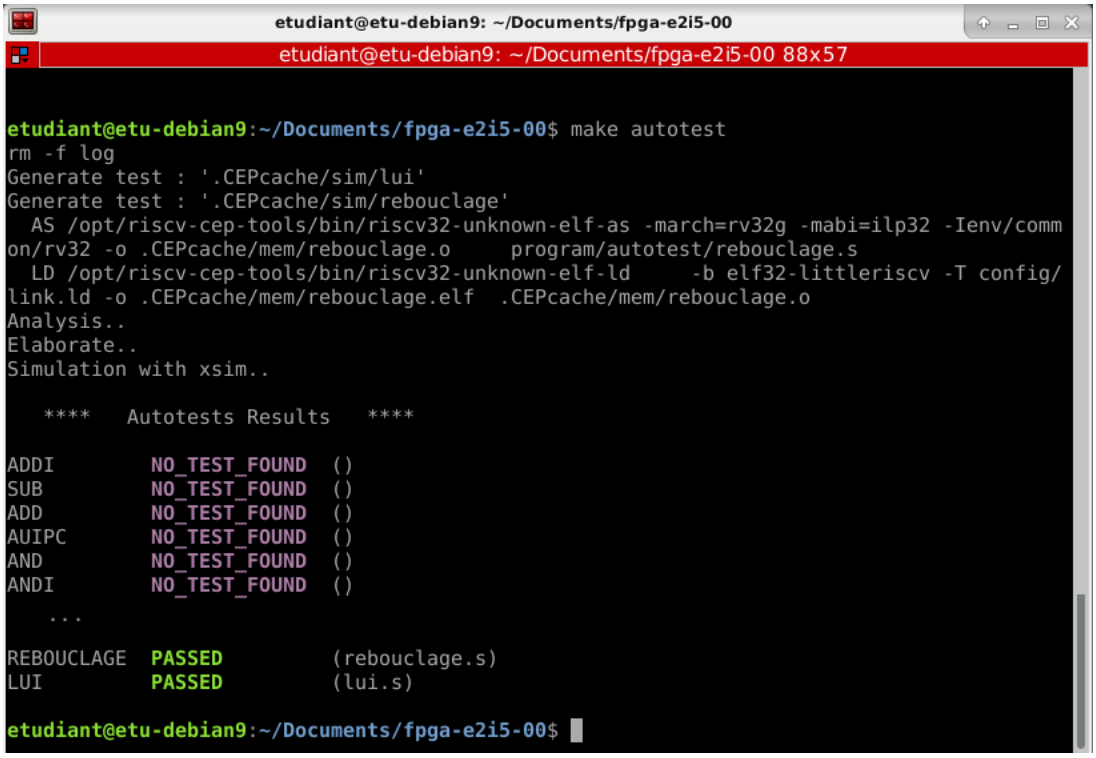

Figure 3 – Validation des résultats de l'instruction `lui` par autotest

— Une fois l'instruction validée, vous pouvez continuer avec l'instruction suivante.
N'oubliez pas de faire un commit de votre instruction !