Compte-rendu final du projet
Architecture des ordinateurs

Gaëtan CHAMBRES Chrystelle PETUREAU

10/05/2015

Exercice 1 :

Question 1 :

On nous demande de faire en sorte que toutes les opérations de type "iXX" soit confondu avec leur consoeurs. Pour cela, nous avons renommer l'identifiant de l'opérateur I-ALUI en I-FREE1 et en précisant qu'on défini maintenant I-ALUI comme la même chose que I-ALU avec la ligne(isa.h, ligne 29 et 33) :

```
#define I_ALUI I_ALU
```

De ce fait dans isa.c, on commente la partie (isa.c de la ligne 925 à la ligne 951)

```
case I_ALUI:
```

pour ne pas avoir d'erreur lors de la compilation, vu que maintenant I-ALU et I-ALUI sont confondu.

Question 2 :

Version séquentielle

Pour permettre l'utilisation de I-OPL et OPL, voici les modifications apportées dans les fichiers HCL :
Pour l'architecture seq, dans le ficher seq-std.hcl, aux lignes 129 et 130, nous avons ajouté ces deux lignes, qui permettent de vérifier avant l'étage EXECUTE si le registre srcA de l'opération OPL vaut rA ou RNONE. Si le registre vaut RNONE, alors on lit la valeur de la constante valC, sinon on lit valA, le contenu de rA.

```
icode in {OPL} && rA == RNONE : valC;
icode in { OPL } : valA;
```

Nous avons alors testé,après recompilation, avec le code suivant :

```
        .pos 0

        irmovl 5,%eax
        irmovl 5,%ebx
        iaddl 5,%eax
        iaddl 5,%ebx
        addl %ebx,%eax

        halt
```

Et nous avons obtenu le résultat suivant :

FIGURE 1 – Etape 1

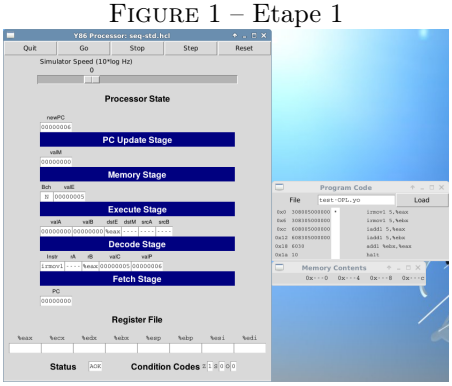


FIGURE 2 – Etape 2 et 3

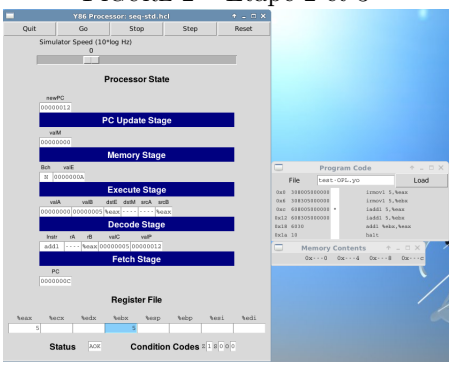


FIGURE 3 – Etape 4 et 5

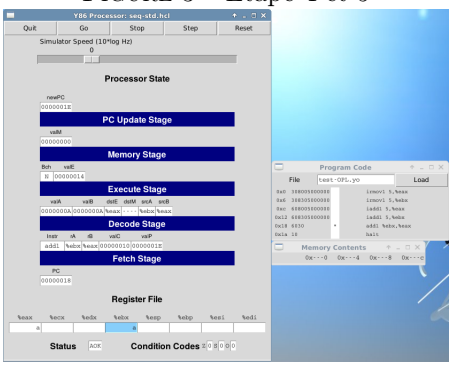
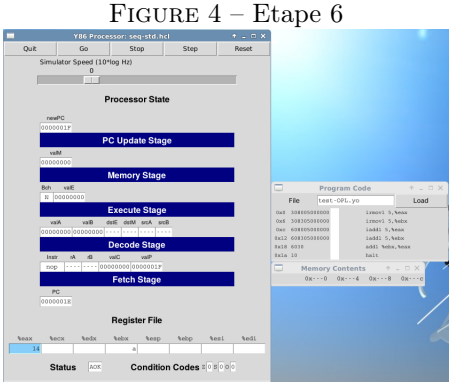


FIGURE 4 – Etape 6



Version pipelinée

Pour l'architecture pipelinée, nous avons ajouté les lignes suivantes (pipe-std.hcl, ligne 204), qui devrait permettre de faire la même vérification que pour la version séquentielle détaillée ci-dessus.

```
E_icode in {OPL} && D_rA == RNONE : E_valC;
        E_icode in {OPL} : E_valA;
```

Mais lorsque nous testons le processeur avec le code suivant, qui est une version adaptée au pipeline du test utilisé précédemment :

```
        .pos 0
        irmovl 5,%eax
        nop
        irmovl 5,%ebx
        nop
        iaddl 5,%eax
        nop
        iaddl 5,%ebx
        nop
        nop
        nop
        nop
        addl %ebx,%eax
        halt
```

les résultats ne sont pas corrects. Le surnombre d'opérateurs NOPE nous permet d'être sur que le problème ne vient pas de bulles manquantes. L'erreur se présente au moment de l'opération "addl" pour laquelle le processeur lit le contenu des deux registres, mais aussi une constante valC égale à 16 (en décimal). Puis à l'exécution, le processeur ajoute au registre B le contenu de cette constante

plutôt que celui du registre A.
Nous n'avons pas su localiser ce problème et donc le résoudre, cependant voici nos résultats :

FIGURE 5 – Etape 1

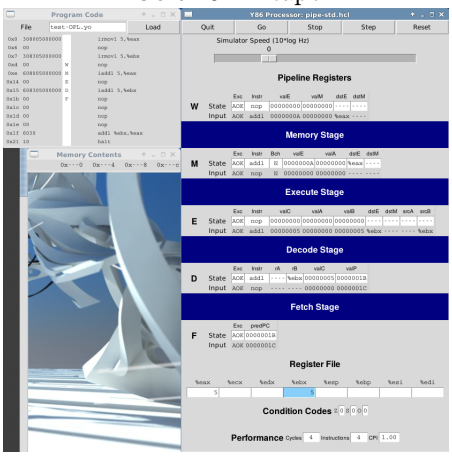


FIGURE 6 – Etape 2

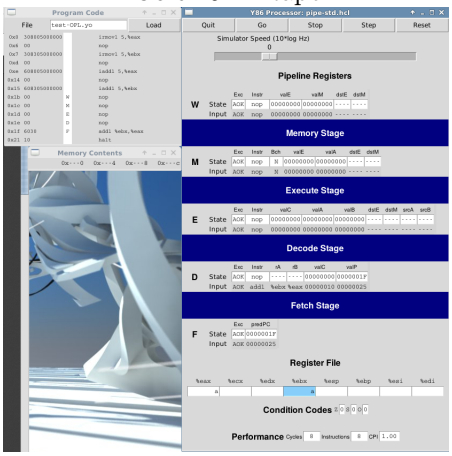
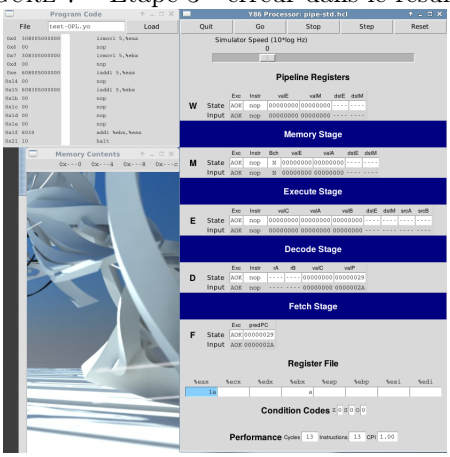


FIGURE 7 – Etape 3 - erreur dans le résultat



Question 3 :

Partie bonus :

Exercice 2 :

On cherche à créer un pas à pas dans le fonctionnement du processeur.

Version séquentielle :

Dans le fichier seq-std.hcl, ligne 92, nous avons ajouté une instruction pour préciser pour quelles valeurs le processeur passera à l'instruction suivante.

```
int instr_next_ifun=[
        1:-1;
        ];
```

Dans le fichier ssim.c à la ligne 375, nous avons ajouter le prototype de la fonction "gen-instr-next-ifun" qui "lit" le compteur d'instructions. On ajoute cette fonction à la ligne 667.

```
if(gen_instr_next_ifun () != -1)
        ifun = gen_instr_next_ifun();
else
```

Enfin à la ligne 772, on ajoute l'instruction :

```
if (gen_instr_next_ifun() == -1){
        pc_in = gen_new_pc();
}
```

Elle permet de calculer la nouvelle valeur du compteur ordinal.

version pipe-linée :

Les modifications sont quasiment similaire à la version séquentielle. Dans le fichier pipe-std.hcl, ligne 138, nous avons ajouté une instruction pour préciser pour quelles valeurs le processeur passera à l'instruction suivante.

```
int instr-next-ifun=[
        1:-1;
        ];
```

Dans le fichier psim.c à la ligne 1327, nous avons ajouter le prototype de la fonction "gen-instr-next-ifun" qui "lit" le compteur d'instructions. On ajoute cette fonction à la ligne 1361.

```
if(gen_instr_next_ifun () != -1)
        if_id_next->ifun = gen_instr_next_ifun();
        fetch_ok= TRUE;
else
        fetch_ok=get_byte_val(mem, valp , &instr );
```

Enfin à la ligne 1388, on ajoute l'instruction :

```
if (gen_instr_next_ifun() == -1){
        pc_next->pc=gen_new_F_predPC();
}
```

Elle permet de calculer la nouvelle valeur du compteur ordinal.

Exercice 3 :

Question 1 :

Question 2 :

Instruction mul :

Instructions lods/stos/movs :

Instruction repstos :

Question 3 :

Question 4 :