Projet de travaux pratiques

Réalisation des processeurs Nono-1 et Nono-2

Le projet est à effectuer en binôme (ou exceptionnellement, en monôme). Le rapport complet **imprimé** est à déposer dans la boîte aux lettres de l'enseignant encadrant la séance de travaux pratiques. Le fichier *Logisim* commenté ainsi que le rapport de projet au format PDF sont à déposer *impérativement* sur Madoc sous la forme d'une archive [1] dont le nom sera composé des noms de chacun des étudiants du binôme — ex. `tartempion-dupont.tar.gz`. La décompression de cette archive devra créer un répertoire `tartempion-dupont/` contenant les fichiers. On re-précisera en commentaire dans le fichier Logisim les noms et prénoms de chacun des étudiants.

Un projet rendu après la **date limite fixée au jeudi 18 décembre 2014, 23h55**, sera affecté d'un malus proportionnel au nombre de jours de retard.

Un processeur à architecture Nono-1 possède 16 registres de 8 bits $R_0, \ldots, R_{15}$ pouvant contenir des entiers signés en complément à 2 et un registre `PC` (*Program Counter*) de 8 bits. Un Nono-1 offre aussi une mémoire de 256 mots de 16 bits pour stocker le code à exécuter.
Le jeu d'instructions supportées par le Nono-1 est présenté dans la table 1. Les trois formats d'instructions F_1, F_2 et F_3 sont décrits dans la figure 1. Toutes les instructions du Nono-1 s'exécutent en un seul cycle.
Le but du projet est de réaliser avec Logisim une implémentation du Nono-1 tel que décrit dans la figure 2, puis de l'utiliser pour exécuter un programme.
On utilisera au mieux les possibilités de création de sous-circuits de Logisim ; on tirera aussi parti des circuits fournis en standard par Logisim (multiplexeurs, ...). Le rapport de projet devra justifier tous les circuits créés (tables de vérités, ...). Pour chaque unité fonctionnelle développée, on créera un sous-circuit chargé de son test afin d'en valider le *design*.

1 Implémentation du Nono-1

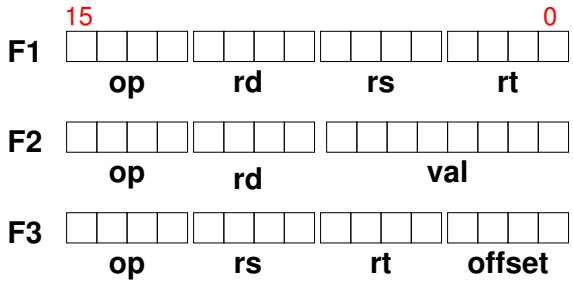

FIGURE 1 – Formats d'instructions

1. Réaliser l'Unité Arithmétique et Logique (UAL) offrant les opérations suivantes sur 8 bits : addition, soustraction, OU bit-à-bit, ET bit-à-bit, NON bit-à-bit, décalage à gauche, décalage à droite logique. L'UAL devra retourner un résultat sur 8 bits ainsi que 4 indicateurs : `CF`, `ZF`, `SF`, `OF`.

   On pourra partir de l'additionneur 4 bits mis au point lors de la première séance de travaux pratiques ou utiliser l'additionneur fourni par Logisim ;

2. Le module « *décodeur d'instruction* » prend en entrée l'opcode de l'instruction courante et retourne quatre indicateurs :

1. Manipulation d'archives :
   — Création de l'archive `titi.tar.gz` à partir du répertoire `titi` : `tar -vzcf titi.tar.gz titi/`
   — Récupération du contenu de l'archive `titi.tar.gz` : `tar -vzxf titi.tar.gz`

Instruction	Format	Commentaire
`add` r_d, r_s, r_t	F_1	Addition
`sub` r_d, r_s, r_t	F_1	Soustraction
`or` r_d, r_s, r_t	F_1	OU bit-à-bit
`and` r_d, r_s, r_t	F_1	ET bit-à-bit
`not` r_d, r_s	F_1	NON bit-à-bit
`shl` r_d, r_s, r_t	F_1	Décalage à gauche logique
`shr` r_d, r_s, r_t	F_1	Décalage à droite logique
`li` r_d, *val*	F_2	Chargement d'un immédiat
`halt`	F_1	Arrêt du programme
`b` *offset*	F_3	Saut inconditionnel *offset* octets en avant ou en arrière
`beq` r_s, r_t, *offset*	F_3	Saut conditionnel *offset* octets en avant ou en arrière
`bne` r_s, r_t, *offset*	F_3	Saut conditionnel *offset* octets en avant ou en arrière
`bge` r_s, r_t, *offset*	F_3	Saut conditionnel *offset* octets en avant ou en arrière
`ble` r_s, r_t, *offset*	F_3	Saut conditionnel *offset* octets en avant ou en arrière
`bgt` r_s, r_t, *offset*	F_3	Saut conditionnel *offset* octets en avant ou en arrière
`blt` r_s, r_t, *offset*	F_3	Saut conditionnel *offset* octets en avant ou en arrière

TABLE 1 – Jeu d'instructions du Nono-1

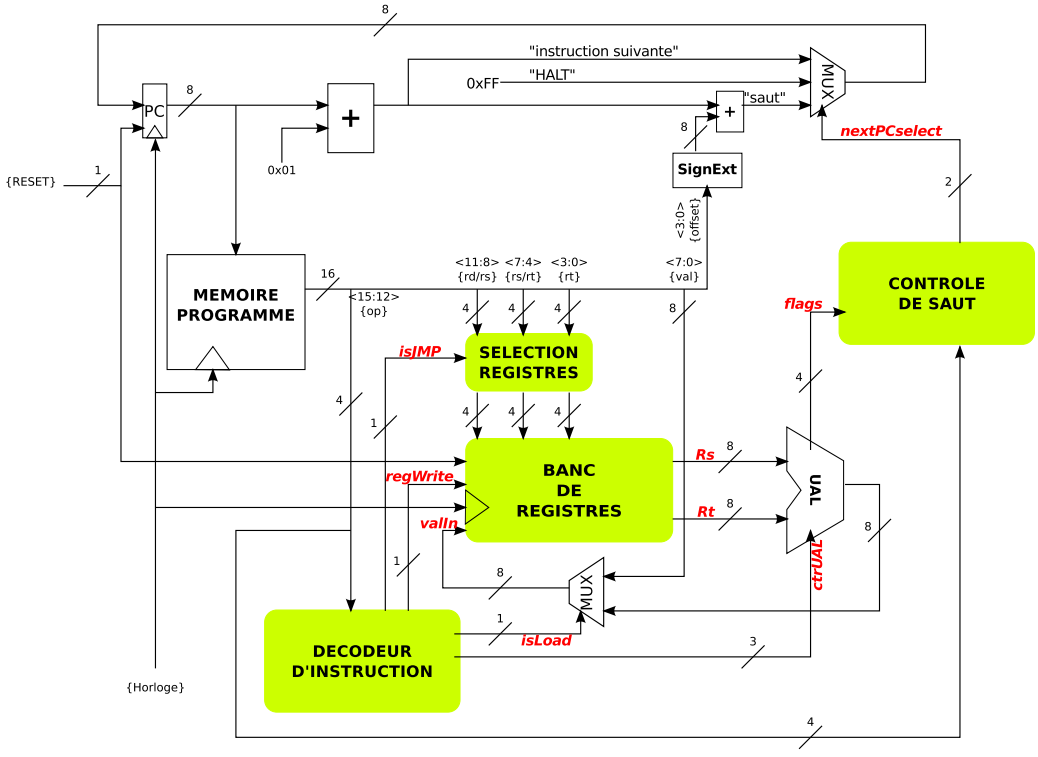

FIGURE 2 – Architecture du Nono-1

`isJMP`. Indicateur valant `1` si l'instruction courante est une instruction de saut ;

`regWrite`. Indicateur valant `1` si l'instruction courante accède à un registre en écriture ;

`isLoad`. Indicateur valant `1` si l'instruction courante est une instruction de chargement ;

`ctrUAL`. Indicateur codant le type d'opération demandée à l'UAL.

Calculer la table de vérité du *décodeur d'instruction*. En déduire son implémentation dans Logisim ;

3. Le module « *contrôle de saut* » prend en entrées l'opcode de l'instruction courante ainsi que les indicateurs `CF`, `ZF`, `SF` et `OF` mis à jour par l'UAL lors de la dernière opération effectuée. Il retourne en sortie un indicateur sur deux bits déterminant la valeur du registre `PC` pour le prochain cycle.
   Écrire la table de vérité du module. En déduire son implémentation dans Logisim ;
4. Le module « *banc de registres* » contient les 16 registres de 8 bits. Il comporte 6 entrées :

   `rd`, `rs`, `rt`. Les indices correspondant, respectivement, au registre à accéder en écriture et aux deux registres à accéder en lecture ;

   `RESET`. Force tous les registres à `0` de manière asynchrone ;

   `regWrite`. Autorise l'écriture dans le registre `R[rd]` ;

   `valIn`. Un entier sur 8 bits à stocker dans `R[rd]` ;

   et deux sorties `Rs` et `Rd`, correspondant aux valeurs des registres `R[rs]` et `R[rd]`.
   Implémenter le banc de registres dans Logisim.
5. Le module « *sélection de registres* » reçoit en entrée 3 champs de 4 bits de l'instruction courante (voir fig. 2) ainsi que l'indicateur `isJMP`, et il retourne les indices correspondant à `rd`, `rs` et `rt` en fonction du format de l'instruction courante.
   Implémenter le module *sélection de registres* ;
6. Implémenter le circuit complet du Nono-1 en réutilisant les sous-circuits développés dans les questions précédentes.

2 Utilisation du Nono-1

On va désormais utiliser le Nono-1 pour exécuter des programmes écrits en code machine Nono-1.

1. D'après la figure 2, quelle est la méthode utilisée pour implémenter l'instruction `halt` ? Quelle contrainte impose t'elle sur les programmes en code machine exécutés ?
2. Traduire en assembleur Nono-1 puis en code machine le programme ci-dessous :

```
1  /* Calcul du pgcd de i et j. */
2   int i, int j;
3   while (i != j) {
4     if (i > j) {
5         i -= j;
6     } else
7         j -= i;
8   }
9   /* Le pgcd de i et j apparaît dans ces deux variables. */
```

   Charger le programme dans la mémoire du Nono-1 et l'exécuter ;
3. Proposer un autre programme non trivial tirant parti du jeu d'instructions du Nono-1. Le traduire en assembleur Nono-1 et en code machine. Sauver ce code machine dans un fichier *via* Logisim et fournir ce fichier dans l'archive rendue ;
4. Le processeur Nono-2 est basé sur le Nono-1 mais il offre la possibilité de programmer des routines (fonctions). Proposer une modification de l'architecture du Nono-1 pour obtenir le Nono-2 et l'implémenter.