TP Procesim

ALM2 RICM3 – simulation de processeur

1 Introduction

Objectifs du TP :

- comprendre l'interprétation des instructions par le processeur ;
- comprendre le microcode en lien avec le graphe de la PC vu en TD.

Pour cela, munissez-vous de la documentation PROCESIM qui a été distribuée en TD. Les fichiers correspondant à cette documentation et aux exemples mentionnés dans cet énoncé sont disponibles dans moodle.

2 Travail demandé

Vous devrez rendre un compte-rendu par binôme à l'issue de l'ensemble des séances de travail autour de Procesim. Votre compte-rendu est à rendre sur moodle avant le lundi 24 mars 2014 à 23h59. Il devra impérativement être au format PDF et nommé `CR_NOM1_NOM2.pdf` où `NOM1` et `NOM2` sont remplacés par les noms (pas les prénoms) des deux personnes du binôme. Il devra être constitué :

- des réponses aux questions de la partie 3 ;
- d'une description textuelle et du listing commenté du microcode que vous aurez écrit pour chaque amélioration demandée (parties 4 et 5) ;
- d'une description de votre stratégie d'élaboration du jeu de programmes de test (parties 4 et 5) ;
- du contenu commenté de chacun des programmes de test que vous aurez écrit en précisant : l'objectif du test, le résultat attendu, le résultat obtenu et la raison (s'il est différent du résultat attendu).

Vous devrez déposer sur moodle (un dépôt par binôme), une archive (au format `tar.gz`) contenant :

- le compte-rendu tel que décrit ci-dessus ;
- le code source de votre microcode (fichier `.muc`) ;
- vos fichiers de tests (sources `.s` et « compilés » `.ob`).

3 Première partie : prise en main et observation

Observez le déroulement d'un programme complet, instruction par instruction, à l'aide des exemples (it et no_it) fournis.

Questions sur le code des exemples :

1. Que font les instructions du traitant de ré-initialisation (*reset* de no_it) et comment est effectué le branchement au début du programme utilisateur ?
2. Choisissez parmi les réponses suivantes le niveau auquel est effectuée l'initialisation du pointeur de pile SP (exemple no_it). Retranscrivez les lignes correspondantes de la partie qui s'en occupe.
   (a) au niveau du matériel du processeur (microcode) ;
   (b) au niveau du programme principal ;
   (c) au niveau du traitant d'interruption.

3. Dans l'exemple it, quelle partie du programme empile-t-elle des octets ? Comment y accède-t-on ?
4. À quelle adresse est stocké le premier octet empilé ?

Questions sur la PC du processeur (le microcode) :

1. Que se passe-t-il au cours de la première microinstruction exécutée après la mise sous tension ou la réinitialisation du processeur. Quels registres (ou parties de registres) sont initialisées ?
2. Combien de microactions sont exécutées entre deux passages à fetch ?
   (a) pour une instruction de calcul reg op= reg
   (b) pour une instruction de calcul reg op= #imm
   (c) pour une instruction blcond

4 Seconde partie : load et store

Complétez le microcode fourni pour ajouter la prise en charge des instructions *load/store* et écrivez un ou plusieurs programme(s) pertinent(s) pour tester votre implémentation sur les variantes de *load* et *store*.

5 Troisième partie : interruptions

Complétez le microcode fourni pour gérer les interruptions. Dans notre processeur, une interruption devra provoquer l'exécution de la séquence de micro-actions suivante :

```
slr <- pc
spsr <- cpsr
masquage des it (bit I)
pc <- adresse vecteur
```

Notre processeur est dépourvu de mode d'exécution (tous les registres sont toujours visibles).

Dans notre processeur nous avons choisi les vecteurs d'interruptions suivants :

- reset : 0
- interruption matérielle : 2
- interruption logicielle : 4

Interruptions matérielles : Pour gérer complètement les interruptions matérielles, il faudra :

- tester la présence du signal d'interruption et l'absence de masquage avant le fetch et provoquer, le cas échéant, le départ en interruption
- implémenter l'instruction rti du jeu d'instructions

Observez le déroulement d'une interruption externe (IRQ) à l'aide de l'exemple it. Vérifier l'impact de `accept/ignore` sur la prise en compte des interruptions et écrivez un ou plusieurs programme(s) supplémentaires pour tester le bon fonctionnement de votre implémentation.

Interruptions logicielles : Afin de permettre à notre processeur de gérer les interruptions logicielles, choisissez un codage pour l'instruction `swi` (interruption logicielle / appel au superviseur). Modifiez alors le microcode pour ajouter la prise en charge de `swi` et écrivez le ou les programme(s) d'exemple correspondant pour tester le bon fonctionnement de votre implémentation.