Université Bordeaux 1, Sciences et technologies
UFR Informatique et Mathématiques

Licence d'informatique
Session de janvier 2002

Architecture et Microprocesseurs (AMP)
Le 22 janvier 2002

Durée : trois heures

Tous documents autorisés sauf photocopies illégales de livres. Votre voisin n'est pas un document.

Rappel : Soyez bref et clair ! Personne n'est impressionné par la quantité !

Bon courage !

Exercice 1. (7p)

Dans une application de type chronomètre, on souhaite réaliser un circuit séquentiel similaire à un compteur. Ce circuit doit avoir deux entrées (e_1 et e_0) et deux sorties (s_1 et s_0). Chaque entrée est connectée à un bouton, et lorsqu'on appuie sur le bouton, l'entrée correspondante prend la valeur 1. Lorsque le bouton est relâché, l'entrée correspondante prend à nouveau la valeur 0. On suppose qu'il est possible d'appuyer sur au plus un bouton à la fois.

Si l'utilisateur de ce circuit appuie (même temporairement) sur le bouton correspondant à l'entrée e_1, le compteur compte de manière cyclique : `00`, `01`, `10`, `11`, `00`, ... Si par contre il appuie (même temporairement) sur le bouton correspondant à l'entrée e_0, le compteur cesse de compter et maintient sa valeur. Si l'utilisateur appuie à nouveau (même temporairement) sur le bouton correspondant à l'entrée e_1, le compteur compte à nouveau à partir de son état précédent.

(a) **[4p]** Donner la table d'états du circuit (indication : vous avez besoin d'une sortie invisible pour déterminer si le compteur doit compter ou non).

(b) **[3p]** Réaliser le circuit correspondant au bistable s_0.

Exercice 2. (13p)

Souvent, dans un processeur moderne, la mémoire centrale est beaucoup plus lente que les registres. Le protocole d'appel habituel (utilisant la pile) doit empiler l'adresse de retour, et accède donc à la mémoire centrale. Ceci peut induire une inefficacité considérable pour les sous programmes n'ayant pas besoin d'appeler d'autres sous programmes (on appelle ce type de sous programmes *feuilles*). En effet, si le traitement utile d'une telle feuille est modeste, le protocole d'appel peut prendre une fraction considérable de son temps d'exécution.

Pour améliorer la situation, on souhaite introduire deux nouvelles instructions, JSRR (JSR Register) et RETR (RET Register). Au lieu d'empiler l'adresse de retour comme JSR, JSRR la stocke dans un registre que nous allons appeler RAR (Return Address Register), puis transfère le contrôle à la sous routine dont l'adresse est donnée en argument. RETR fait l'inverse, à savoir qu'il transfère le contrôle à l'adresse stockée dans RAR. Une autre instruction PUSHR est nécessaire pour empiler RAR. L'ancienne instruction JSR est maintenant équivalente à une JSRR suivie d'une PUSHR. L'appelant se sert uniquement de l'instruction JSRR. Si l'appelé est une feuille, il se sert de l'instruction RETR. Sinon, il doit émettre une instruction PUSHR avant de commencer sont traitement habituel. Il utilise alors l'instruction RET pour retourner dans l'appelant.

(a) [**4p**] Rajouter les circuit nécessaires pour la réalisation des instructions JSRR, RETR et PUSHR à l'architecture donnée en annexe.

(b) [**3p**] Réaliser le microprogramme des instructions JSRR et RETR.

(c) [**6p**] Donner le code généré par le compilateur pour l'architecture donnée en annexe pour les deux sous programmes suivants :

```
int f(int x)
{
  ... /* aucun appel de fonction ici */
}

int g(void)
{
  f(123);
}
```