Université Bordeaux 1, Sciences et technologies
UFR Informatique et Mathématiques

Licence d'informatique
Session de janvier 2003

Architecture et Microprocesseurs (AMP)
Le 20 janvier 2003

Durée : trois heures

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

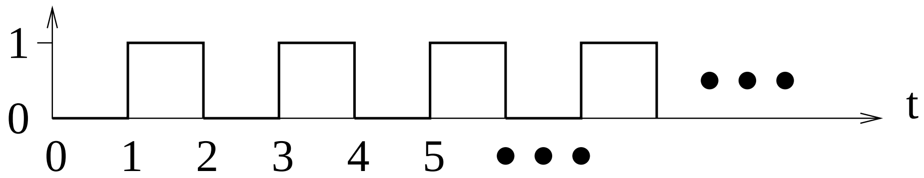
Rappel : Soyez bref et clair ! Personne n'est impressionné par la quantité !

Bon courage !

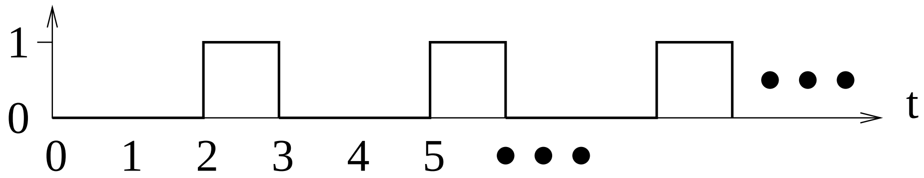
Exercice 1. (7p)

Dans un outil pour tester des circuits numériques, nous avons besoin d'un générateur de signaux binaires.

Dans notre cas, nous avons besoin d'un circuit séquentiel capable de générer deux types de signaux. Le premier signal est illustré par la figure suivante :



Le deuxième signal est illustré par la figure suivante :



Pour passer d'une signal à l'autre, on dispose de deux boutons chacun connecté à un signal d'entrée qui vaut 1 si on appuie sur le bouton correspondant, et 0 sinon. Lorsque l'utilisateur appuie (même momentanément) sur le premier bouton (signal $b1$), le circuit doit générer le premier signal. Lorsque l'utilisateur appuie (même momentanément) sur le deuxième bouton (signal $b2$), le circuit doit générer le deuxième signal. On suppose qu'il est mécaniquement impossible d'appuyer sur les deux boutons simultanément.

Pour la réalisation de ce circuit, vous avez besoin de deux sorties invisibles. La première ($s1$) sert à déterminer lequel entre les deux signaux doit être généré (0 = le premier signal, 1 = le deuxième signal). La deuxième ($s2$) sert uniquement pour le deuxième signal, et détermine depuis combien de temps (0 = inférieur à une période d'horloge, 1 = supérieur à une période d'horloge) il vaut 0.

a. (4p) Remplir la table d'états suivante (la sortie s est la sortie du signal à générer) :

s	b1	b2	s1	s2	s′ s1′ s2′
0	0	0	0	0	
0	0	0	0	1	
0	0	0	1	0	
0	0	0	1	1	
0	0	1	0	0	
0	0	1	0	1	
0	0	1	1	0	
0	0	1	1	1	
0	1	0	0	0	
0	1	0	0	1	
0	1	0	1	0	
0	1	0	1	1	
0	1	1	0	0	
0	1	1	0	1	
0	1	1	1	0	
0	1	1	1	1	
1	0	0	0	0	
1	0	0	0	1	
1	0	0	1	0	
1	0	0	1	1	
1	0	1	0	0	
1	0	1	0	1	
1	0	1	1	0	
1	0	1	1	1	
1	1	0	0	0	
1	1	0	0	1	
1	1	0	1	0	
1	1	0	1	1	
1	1	1	0	0	
1	1	1	0	1	
1	1	1	1	0	
1	1	1	1	1	

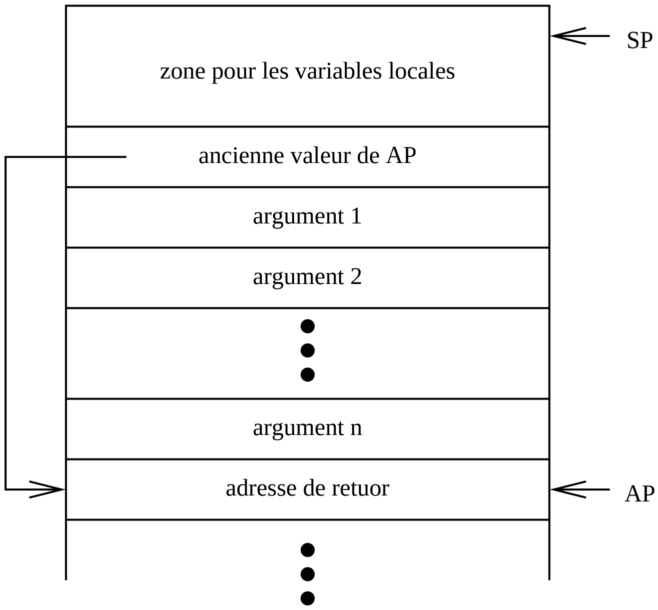
(b) **[3p]** Réaliser le circuit correspondant au bistable s_2.

Exercice 2. (13p)

Un inconvénient du protocole d'appel de fonctions habituellement utilisé pour le langage C est que l'appelé ignore le nombre d'arguments que l'appelant a passé. Il est alors nécessaire pour l'appelant de dépiler les arguments après l'appel.

Dans certains langages de programmation, il est nécessaire pour l'appelé de connaître cette information et de pouvoir dépiler les arguments à la place de l'appelant. Une solution du problème est de rajouter un registre que nous allons appeler *pointeur d'arguments* où AP et qui marque la fin de la liste des arguments.

De plus, pour que l'appelé puisse dépiler les arguments sans affecter l'adresse de retour, cette dernière sera "plus bas" dans la pile (i.e., vers une adresse plus haute en mémoire) que les arguments. Avec ces modifications, voici la nouvelle disposition de la pile pendant l'exécution de l'appelé (juste après allocation de la place pour des variables locales).



Pour le nouveau protocole d'appel, nous avons besoin de cinq nouvelles instructions : PUSHAP, POPAP, INITAP, JSRAP et WIPE.

L'instruction PUSHAP empile le contenu actuel du nouveau registre AP. C'est la première instruction exécuté par l'appelé. Un appelé *feuille* (à savoir que n'appelle aucune autre fonction à son tour) n'a pas besoin d'exécuter cette instruction.

L'instruction POPAP dépile le sommet de la pile et met la valeur dans AP. Cette instruction doit être utilisé par un appelé (non feuille) avant de retourner dans l'appelant (après désallocation des variables locales).

Avant de commencer le calcul des arguments à passer, l'appelant doit exécuter l'instruction INITAP dont l'effet est le suivant :

1. décrémenter SP pour laisser la place pour l'adresse de retour, et
2. copier la valeur de SP dans AP pour marquer la fin de la liste des arguments.

Pour effectuer l'appel, l'appelant se sert de la nouvelle instruction JS-RAP, similaire à JSR. Son effet et de stocker le contenu de PC dans la à la position indiquée par AP et de sauter à l'adresse indiqué par l'argument de l'instruction.

L'appelé, juste avant de retourner dans l'appelant, exécute la nouvelle instruction WIPE dont l'effet est de copier le contenu de AP dans SP, ce qui supprime les arguments. Finalement, il se sert de l'instruction RET habituelle pour retourner dans l'appelant.

(a) [**4p**] Rajouter les circuits nécessaires pour la réalisation des instructions PUSHAP, POPAP, INITAP, JSRAP et WIPE à l'architecture donnée en annexe.

(b) [**3p**] Réaliser le microprogramme des instructions INITAP, JSRAP et WIPE.

(c) [**6p**] Donner le code généré par le compilateur pour l'architecture donnée en annexe (augmenté des nouveaux circuits et des nouvelles instructions) pour les deux sous programmes suivants :

```
void f(int x, int y)
{
  int z;
  ...
}

int g(void)
{
  f(123, 456);
}
```