TP3

Programmation en assembleur sur CRAPS

Interruptions – BE commutation

1- Les entrée sorties

Le simulateur craps dispose :

- d'une sortie sous la forme de 16 leds (16 petits ronds en dessous de NZVC, qui sont mappés à l'adresse 0xB0000000. La valeur écrite sur les 16 bits de faible poids à cette adresse, est affichée sur les 16 leds. La séquence suivante affiche les 16 bits de faible poids de r1 sur les leds :

```
LEDS = 0xB0000000
set     LEDS, %r2
st      %r1, [%r2]      //écriture du contenu de r1 à l'adresse 0xB0000000
```

- d'une entrée sous la forme de switches sur 16 bits, mappée à l'adresse 0x90000000.

2- Les Interruptions

Le simulateur CRAPS implante pour le moment une seule interruption : Une IT manuelle qui a lieu à chaque appui sur le bouton IT (en haut à droite, au dessus de NZVC), et qui interrompt le programme en cours d'exécution pour donner la main au « handler » (sous-programme) d'interruption.

Pour préserver le contexte du programme interrompu, l'adresse à laquelle a eu lieu l'interruption et les indicateurs NZVC (registre d'état étendu à un mot de 32 bits) sont mémorisés dans la pile.

Le handler d'interruption se termine par l'instruction ***reti*** qui dépile les indicateurs NZVC et l'adresse, pour redonner la main au programme interrompu.

Pour des raisons de simplicité, le handler est mis par défaut à l'adresse 1.
Donc, un programme assembleur CRAPS avec IT a la forme suivante :

```
                ba      debut_prog
handler :       ....            // handler à l'adresse 1
                ....
                reti            // retour d'IT
debut_prog : …
```

Illustration : Dans l'exemple suivant (à tester), à chaque interruption, %r1 est incrémenté.

```
                ba      Debt_prog
Handler :
                inc     %r1
                reti

Debut_prog :    …
Bcle_prog :     nop
                ba      Bcle_prog
Compteur :      .word   0
```

2- BE commutation

Le but est de réaliser une version simplifiée, et manuelle, de la commutation entre plusieurs programmes (processus) qui s'exécutent en pseudo-parallélisme.
On utilisera l'interruption manuelle IT, ce qui permettra de mieux maitriser les tests, et on adoptera les choix suivants :

- Un seul programme servira pour tous les processus, dont on limitera le nombre max à 15
- Ce programme pourra utiliser les registres r1, r2, … jusqu'à r9. Il connaît son numéro qui reste mémorisé dans le registre %r1, et gère un compteur temps qui est mémorisé dans le registre %r2
- Ce programme exécute l'algorithme suivant :

  (1) Répéter indéfiniment

  Afficher son numéro (r1) sur leds[7..0]
  Afficher son compteur (r2) sur leds[15..8]
  Temporiser durant 1 seconde environ
  Eteindre leds[7..0] tout en gardant le compteur (r2) sur leds[15..8]
  Temporiser durant 1 seconde environ
  Incrémenter son compteur

  Ceci nous permettra de vérifier visuellement que le processus retrouve bien son contexte (numéro et compteur) quand il reprend la main.
- Un programme principal effectuera les initialisations nécessaires et se mettra à boucler jusqu'à ce qu'il soit interrompu par la première interruption qui donnera la main au processus 1.
- La main sera donnée aux différents processus dans un ordre croissant

Il est conseillé de procéder par étapes selon l'ordre suivant :

1- Ecrire et tester le programme « proc » qui implante l'algorithme (1) présenté plus haut.

2- Ecrire le handler d'interruption, qui se contente, dans un premier temps, d'incrémenter le registre r1 modulo NB_PROC (NB_PROC=3 dans un 1er temps). Le résultat ressemble au produit final (sauf pour le compteur), mais il n'y a aucune commutation pour l'instant, car le retour se fait toujours dans le même processus (en mettant un point d'arrête à l'entrée dans le handler et un autre sur le reti, vérifier que l'adresse d'interruption dans la pile.

3- Déclarer un tableau « Tab_sp » initialisé avec les adresses de 16 piles différentes, et :
   - ajouter dans le handler les instructions qui permettent de changer de pile,
   - créer un programme d'initialisation qui :
     - met dans la pile de chaque processus ce qui est nécessaire pour que le handler puisse lui donner la main comme il le faut lors de la 1ère entrée.
     - se met à boucler en attendant la première interruption qui le suspendra et donnera la main au processus numéro 1

   Tester en mettant un point d'arrêt à l'entrée du handler et en l'exécutant pas à pas (noter l'adresse d'interruption de chaque processus, et vérifier que l'on revient au même endroit lorsque ce dernier reprend la main).

4- Compléter le handler et le programme d'initialisation pour sauvegarder le contexte complet (tous les registres susceptibles d'être utilisés) de chaque processus.