TD 02 – Interruptions

(TD issu du cours de Jean-Luc Massat)

On va travailler sur une machine très simple : la CPU ne comporte que trois registres spécialisés (le `MODE`, le compteur ordinal `CO` et le pointeur de pile `SP`) et deux registres généraux (`R1` et `R2`). Pour simplifier, le mot d'état du processeur regroupe tous les registres.

$$\texttt{mep} = \langle \texttt{CO,MODE,SP,R1,R2} \rangle$$

On vous rappelle que pour traiter les interruptions, il existe dans la machine un vecteur d'interruptions qui débute à l'adresse `vi`. Ce vecteur contient les adresses des traitants associés à chacune des causes d'interruption (2 causes possibles dans notre cas). On définit les constantes suivantes :

Numéro	Signification	Constante
0	déroutement sur erreur d'adressage	`INT_ERR_ADR`
1	déroutement sur instruction inconnue	`INT_ERR_INSTR`

Lorsqu'une interruption de numéro `k` se produit, le mécanisme (câblé) des interruptions déroule les actions suivantes :

```
mem[SP] := ⟨CO,MODE⟩; // empiler le CO et le MODE
SP := SP + 1;
MODE := Maitre; // passer en mode maître
CO := vi[k]; // se brancher sur le traitant
```

Le gestionnaire d'interruptions offre deux routines permettant à un traitant de sauvegarder et de restaurer facilement la valeur des registres de la CPU :

```
sauver_mep(var m:mep)
début
    SP := SP - 1;
    ⟨m.CO,m.MODE⟩ := mem[SP]
    m.SP := SP
    m.R1 := R1
    m.R2 := R2
fin
```

```
charger_mep(m:mep)
début
    SP := m.SP;
    mem[SP] := ⟨m.CO,m.MODE⟩
    SP := SP+1
    R1 := m.R1
    R2 := m.R2
    RTI
fin
```

L'instruction `RTI` (*Return To Interrupted*) se charge de revenir au code interrompu. Elle est définie de la manière suivante :

```
SP := SP - 1;
⟨CO,MODE⟩ := mem[SP]
```

Attention : on ne revient pas de l'appel à la fonction `charger_mep`.

Exercice 1. *Calculer la taille de la mémoire*

✎ On cherche à écrire l'algorithme permettant de déterminer la taille réelle de la mémoire. Pour cela, nous allons utiliser le déroutement `INT_ERR_ADR`. La mémoire est organisée en barrettes, chaque barrette contient `TAILLE_BANC` mots de 32 bits, et nous ferons l'hypothèse importante que la mémoire est contiguë (c'est-à-dire qu'une fois que vous avez dépassé, vous êtes sûrs qu'il n'y a plus rien après). Ce programme fonctionne en mode maître (il est exécuté au démarrage du système).

Exercice 2. *Simuler des instructions manquantes*

Sur certaines configurations (moins chères que les hauts de gamme), certaines instructions au lieu d'être câblées sont simulées de manière logicielle. Le mécanisme de déroutement est utilisé dans de tels cas de figure pour programmer cette simulation.

1. On vous demande d'écrire le traitant du déroutement `INT_ERR_INSTR` de manière à ce que les deux nouvelles instructions `INC_R1` (incrémenter `R1`) et `INC_R2` (idem) soient simulées correctement.

   Remarque 1. *En cas de déroutement, le compteur ordinal pointe sur le code opération de l'instruction qui a provoqué l'erreur.*

   Remarque 2. *Le traitement des erreurs se résume à l'appel d'une fonction* `erreur()`.

2. En utilisant le mécanisme de la question précédente, on vous demande de simuler l'exécution d'une nouvelle instruction qui s'appelle `SVC`. Cette instruction possède un argument. Suivant la valeur de cet argument (1 ou 2), cette instruction incrémente le registre `R1` ou `R2`. On peut (par exemple) l'utiliser de la manière suivante :

```
LOAD R1, #10; // charger le registre R1 avec 10
SVC 1 // incrémenter le registre R1
```

Exercice 3. *Récupération des erreurs*

En général, un déroutement survenant durant l'exécution d'un processus utilisateur provoque l'envoi d'un message signifiant la cause puis le retour au superviseur qui prend en charge un nouveau processus. Il se peut, dans certain cas, que l'utilisateur désire « récupérer » ce déroutement (c'est la fonction `signal` du C ANSI). Dans ces conditions, si le langage le permet, cela se traduira par un appel au superviseur accompagné de deux paramètres : la cause de déroutement et l'adresse de la procédure utilisateur de récupération.
Attention : cette récupération des erreurs n'est valable qu'une seule fois, si une deuxième erreur se produit le traitement standard sera appliqué.

1. Proposez une programmation de la routine système qui « installe » la fonction utilisateur. Cette routine a le prototype suivant :

   `svc_signal(cause` **: entier,** `fonc` **: pointeur de fonction**`);`

2. Comment modifier le traitement des déroutements pour prendre en compte la fonction utilisateur (si elle est présente) ?