IFT209 –Programmation système

Laboratoire 5

Enseignant: Rosa Garcia
Date de remise: indiquée sur **Turnin**
Modalités de remise: **Turnin**
À réaliser: personne seule ou en équipe de deux

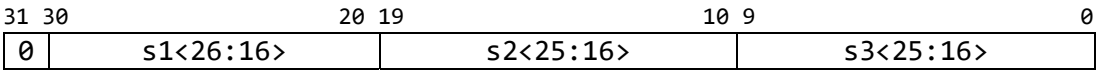
Le but de ce laboratoire est de pratiquer la programmation avec bits et chaînes de bits.

Problème. La génération de nombres pseudo-aléatoires peut être réalisée à l'aide d'algorithmes simples comme celui qu'on retrouve dans la fonction `rand()` de la bibliothèque de C (`stdlib`), qui transforme une valeur initiale nommée *seed* avec des opérations arithmétiques et logiques pour obtenir un nombre pseudo-aléatoire et la valeur initiale suivante qui sera utilisée lors du prochain appel.

Algorithme. La fonction `rand()` dérive trois valeur de la *seed* (*s1*, *s2* et *s3*) à l'aide d'une multiplication avec une constante *m* et d'une addition avec une constante *n*. Elle combine ensuite ces trois valeurs à l'aide d'opérations logiques pour obtenir le résultat *r*. La valeur *seed* est ensuite remplacée par *s3*. Voici le calcul détaillé:

```
s1 := (seed × m) + n
s2 := (s1 × m) + n
s3 := (s2 × m) + n
seed := s3
```

Le résultat *r* est ensuite composé comme suit, en binaire:



- Le bit 31 vaut 0 (les valeurs générées sont exclusivement positives);
- Les bits 20 à 30 inclusivement prennent les valeurs des bits 16 à 26 de *s1*;
- Les bits 10 à 19 inclusivement prennent les valeurs des bits 16 à 25 de *s2*;
- Les bits 0 à 9 inclusivement prennent les valeurs des bits 16 à 25 de *s3*;

Tâche. Vous devez écrire un sous-programme, en langage d'assemblage ARMv8, qui génère nombre pseudo-aléatoire et la prochaine valeur initiale (*seed*) en suivant l'algorithme expliqué plus haut.

Votre sous-programme recevra en paramètre l'adresse de la valeur initiale, la constante multiplicative *m* et la constante additive *n*. Il devra retourner la valeur pseudo-aléatoire *r*, puis modifier la valeur initiale pour la prochaine en écrivant celle-ci à l'adresse fournie en paramètre.

Votre sous-programme doit être écrit dans le fichier `rng.as` et s'appeler `Random`. Il sera compilé avec un programme principal qui effectuera des tests sur son fonctionnement.

Tests. Le programme duquel fera partie votre fonction `Random` vérifie si les nombres générés sont répétitifs (période) et si la distribution de la densité de bits à 1 dans les nombres générés est uniforme (écart-type). Plusieurs fichiers de tests sont fournis sur **Turnin**. Les résultats attendus sont les suivants:

```
      Test                    Résultat

rngtest < test1               Periode: 345000 (maximale)
                              Ecart-type: 55.594296

rngtest < test2               Periode: 1
                              Ecart-type: 4936.448073

rngtest < test3               Periode: 345000 (maximale)
                              Ecart-type: 2466.975976

rngtest < test4               Periode: 345000 (maximale)
                              Ecart-type: 57.230465

rngtest < test5               Periode: 345000 (maximale)
                              Ecart-type: 57.280391
```

Rappels.

- Les décalages logiques à gauche et à droite (`lsl`, `lsr`) servent à déplacer des bits dans un registre;
- Le *et* logique (`and`) peut servir à sélectionner et copier des bits d'un registre vers un autre;
- Le *ou* logique (`orr`) peut servir à copier des bits vers une partie d'un registre contenant des 0.

Directives.

- Votre programme doit être obtenu en complétant le code fourni sur **Turnin**;
- Votre programme doit être remis dans un seul fichier nommé `rng.as`;
- Ne modifiez pas le fichier `rngtest.cc`, l'original sera utilisé pour la correction;
- Supposez que les paramètres reçus par votre sous-programme sont valides.

Pointage. Vous pouvez obtenir jusqu'à 10 points répartis ainsi:

- 5 points si votre programme passe les cinq tests ci-dessus;
- 3 points si votre programme affiche la bonne sortie sur d'autres tests choisis à la correction
- 2 points pour la lisibilité du code (indentation, commentaires, conventions).