UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA - DEPARTAMENTO DE INFORMÁTICA APLICADA

INF01107 - Arquitetura e Organização de Computadores I – Turmas A, B e C - 2005/1

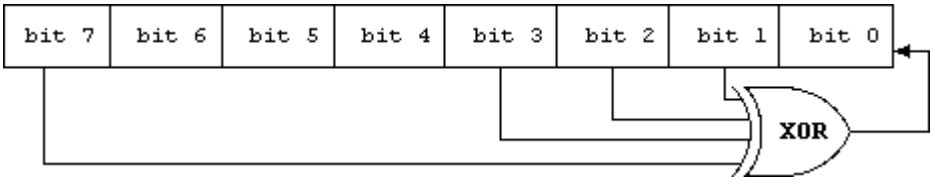
Trabalho Prático 2 - Simulador AHMES

Escrever um programa para o computador Ahmes que implemente um gerador de seqüências pseudo aleatórias de números inteiros positivos, simulando o funcionamento de um dispositivo denominado "registrador de deslocamento com realimentação linear" (LFSR – linear feed-back shift register).

Conceitos básicos de LFSR

Um LFSR é um circuito formado por um registrador de deslocamento e uma porta XOR de 2 ou mais entradas. A cada operação, este circuito desloca o conteúdo do registrador em um bit (para a direita ou para a esquerda) e substitui o bit liberado no deslocamento (o mais significativo ou o menos significativo, conforme o sentido do deslocamento) por um bit calculado através de uma operação "ou exclusivo" (XOR) efetuada sobre dois ou mais bits do valor nele contido antes do deslocamento. Dependendo da forma de implementação, um LFSR de **n** bits pode gerar seqüências de até 2^{n-1} valores distintos, no intervalo $[1,2^{n-1}]$.

A figura a seguir mostra o diagrama esquemático de um LFSR de 8 bits em que o deslocamento ocorre para a esquerda e o bit menos significativo é substituído pelo resultado de um XOR entre o bits 7, 3, 2 e 1 do valor armazenado no LFSR antes do deslocamento. Esta implementação permite gerar seqüências de 255 valores inteiros positivos diferentes, no intervalo [1,255].



Para gerar uma destas seqüências de valores, primeiramente deve ser armazenado no LSFR da figura acima um valor diferente de zero (denominado de "semente"). Feito isto, se o valor contido neste LFSR não for mais alterado (através da carga externa de outro valor de semente), a cada operação do LFSR o valor contido no mesmo mudará para um valor diferente, gerando uma seqüência "pseudo aleatória" de valores inteiros positivos naquele intervalo. Enquanto não forem gerados todos os valores neste intervalo (o que só acontece após 254 operações deste LFSR), nenhum valor repetido é gerado. A seqüência assim gerada é dita pseudo aleatória porque, para um mesmo valor de "semente", sempre é gerada a mesma seqüência de valores. Além disto, após gerados os 254 valores possíveis a partir de um determinado valor de semente, o LFSR passa a repetir toda a seqüência de valores, na mesma ordem, indefinidamente. O valor da semente nunca deve ser igual a zero, porque neste caso o próximo valor gerado sempre é zero (0 XOR 0 = 0) e o LFSR perde sua utilidade.

Como já foi dito, o LFSR mostrado na figura acima gera seqüências de 255 valores diferentes. No entanto, dependendo da quantidade dos bits que participam da operação XOR e da posição destes bits, que podem variar de acordo com a arquitetura do LFSR, algumas implementações podem gerar seqüências mais curtas (com menos de 255 valores distintos).

Especificação do programa a ser desenvolvido

O programa deverá simular a operação de um LFSR de 8 bits, cuja arquitetura e operação serão definidas através dos parâmetros de entrada, que serão valores inteiros positivos, a saber:

- palavra 128 - valor da "semente" (*s*), sendo $1 \leq s \leq 255$
- palavra 129 - quantidade de valores aleatórios a gerar (*n*), sendo $1 \leq n \leq 254$
- palavra 130 - "máscara" que indica quais os bits do LFSR devem ser operados pelo XOR (*m*)
- palavra 131 – sentido do deslocamento do conteúdo do LFSR (*d*): 0 se para a esquerda, 1 se para a direita

Com base nestes parâmetros, o programa deverá inicializar o LFSR com o valor *s* e depois gerar uma seqüência de *n* valores aleatórios, armazenando o último valor gerado – e somente este - na palavra 132 da memória.

A máscara será um valor de 8 bits, no qual bits em 1 representam a posição dos bits do LFSR que devem ser operados pela porta XOR. Portanto, a máscara terá sempre dois ou mais bits iguais a 1. Para simular o LFSR mostrado na figura acima, a máscara seria o valor binário "10001110", indicando que deve ser feito o XOR dos bits 7, 3, 2 e 1 do LFSR, e

o sentido do deslocamento seria 0, indicando deslocamento para a esquerda e inserção do resultado do XOR no bit menos significativo do LFSR.

Tendo em vista os intervalos possíveis para os parâmetros de entrada, para evitar erros na execução o programa deverá começar fazendo uma verificação destes dados, da seguinte maneira:

1. Um LFSR nunca pode receber como semente o valor zero, pois neste caso o próximo valor gerado será sempre zero e o programa, se não testar este caso especial, entrará em "*loop*". Portanto, quando o valor de *s* for igual a zero, o programa deverá colocar zero na palavra 132 e terminar sem fazer mais nada.
2. Se o valor de *n* for igual a zero, o programa deverá apenas copiar o valor da semente (palavra 128) para a palavra 132 e terminar sem fazer mais nada.
3. Os valores de *m* e *d* serão sempre válidos, não sendo necessário fazer nenhuma verificação dos mesmos.

Uma vez feitas estas verificações, o programa deverá gerar *n* valores e armazenar na palavra 132 o último valor gerado.

Dicas

- Como o Ahmes tem apenas um registrador na unidade operacional, embora apenas o último valor gerado deva ser armazenado na palavra 132, é necessário armazenar o valor gerado em cada operação de deslocamento realizada pelo LFSR, pois este valor precisará ser usado para gerar o número seguinte na seqüência.
- Embora o Ahmes não tenha uma instrução do tipo XOR, esta operação pode ser implementada usando as instruções NOT, AND e OR ou lembrando que a operação XOR também é denominada de "função ímpar" (ver livro texto).

Correção e exemplos para testes

Os trabalhos serão corrigidos de forma automática, com **20** conjuntos de parâmetros (s, *n*, *m* e *d*) diferentes. Portanto, devem ser observadas rigorosamente as seguintes especificações:
- o código do programa deve iniciar no endereço 0 da memória
- a primeira instrução executável deve estar no endereço 0
- os endereços dos parâmetros de entrada e do resultado devem ser exatamente os especificados (128 a 132)
- usar para variáveis adicionais os endereços de memória de 133 em diante
- os parâmetros de entrada devem estar inalterados ao término da execução de cada caso de teste

O programa deverá ser desenvolvido com o uso do montador Daedalus, sendo a documentação do trabalho incluída como comentário no código fonte (não fazer relatório à parte). Ver exemplo de documentação na página da disciplina.

O trabalho (composto pelos arquivos **.ahd** - código fonte comentado, **.mem** e **.lst**, todos correspondentes à mesma versão) deverá ser entregue da mesma forma que o trabalho com o Neander (Claroline ou disquete, conforme a turma). Para todos os nomes dos arquivos, utilize a letra inicial do seu primeiro nome, seguida do seu número do cartão de identificação (com 6 dígitos), com a extensão de arquivo correspondente.

Exemplos de seqüências geradas, usando a arquitetura mostrada na figura (usar para testar o programa):

s (pal. 128)	*n* (pal. 129)	m_2 (pal. 130)	*d* (pal. 131)	seqüência gerada (excluindo a "semente")	resultado (pal. 132)
0	7	10001110	0	seqüência de zeros – entra em *loop* se não testar este caso	0
156	0	10001110	0	não deve gerar nenhum número – copiar semente na palavra 132	156
156	7	10001110	0	57, 115, 231, 207, 158, 60, 120	120
1	254	10001110	0	1, 2, 5, 11, 22, 44, 88, ..., 16, 32, 64, 128	128
255	10	10001110	0	254, 252, 249, 242, 228, 200, 144, 33, 66, 133	133
255	7	10001110	1	127, 191, 97, 175, 87, 43, 21	21
255	3	10001000	0	254, 252, 248	248

obs: máscara de $10001110_2 = 142_{10}$, $10001000_2 = 136_{10}$

Datas de Entrega: 29/06/2005 (Turma A) e 30/06/2005 (Turmas B e C)