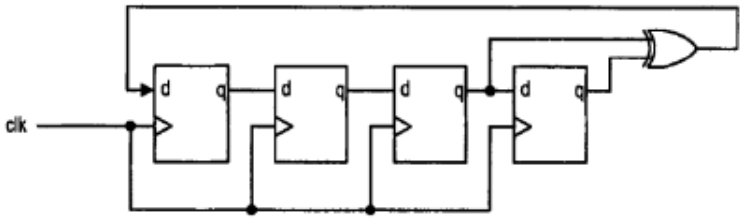
Teste – B1 – 02/09/2021

Pontuação: 3,0 pontos e 25% da frequência

QUESTÃO 1 – Linear Feedback Shift Register (15%)

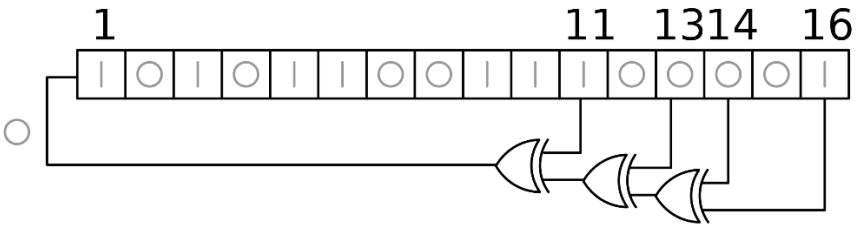
Uma forma simples de se gerar tais números é através de Linear Feedback Shift Registers (LFSRs). Um LFSR nada mais é do que um banco de Flip-Flops D com o mesmo clock, onde a saída de um flip-flop alimenta a entrada do seguinte. A entrada do primeiro flip-flop é gerada através de operações XOR usando o estado dos demais flip-flops. A Figura abaixo mostra um LFSR pequeno, com apenas 4 Flip-Flops:



Note que se, da esquerda para a direita, o estado deste LFSR é 1011, após a batida do clock o estado do LFSR será 0101. Após uma nova batida do clock, o estado será 1010. Um bom LFSR tem uma equação que garante que essa repetição será bastante longa, antes que os estados comecem a se repetir.

Escreva uma função em C que implemente um LFSR com 16 bits de estado com a seguinte equação (também mostrada na figura abaixo):

$$b = x^{16} + x^{14} + x^{13} + x^{11}$$



Ou seja, o bit que vai ser inserido no LFSR é o XOR dos bits nos flip-flops 11, 13, 14 e 16. No exemplo, o estado do LFSR começa com 0xACE1 e, após a batida do clock, passa a ser 0x5670. O bit de saída do canal de realimentação (0, no exemplo) é o bit de resposta. No caso da sua implementação, o clock é a chamada da função – cada vez que a função lfsr for chamada, ela deve calcular o bit de saída, mudar o estado do LFSR, e retornar o bit de saída. Note que o estado do LFSR deve se manter entre chamadas da função (isto é, deve ser declarado como uma variável *global*). A *assinatura* da função **deve** ser:

```
unsigned int lfsr();
```

QUESTÃO 2 – GPIO (55%)

A ideia dessa questão é fazer um jogo Genius com o MSP-430. No jogo Genius, mostrado abaixo, o jogo primeiro mostra uma sequência de teclas piscando os botões coloridos. Em seguida, o jogador deve repetir a sequência mostrada pelo jogo.

Na nossa implementação, temos apenas dois botões – o botão da esquerda (botão P2.1 e LED P1.0) e da direita (botão P1.1 e LED P4.7). Além disso, a comunicação do jogo com o usuário deve ser feita usando os mesmos dois LEDs. Na nossa implementação vamos usar uma variável global:

```
unsigned int sequence[10];
```

para representar a sequência atual. Além disso, sempre que for preciso contar tempo, pode ser utilizada a rotina:

```
void tempo()
{
   volatile unsigned int x = 50000;
   while (x--);
}
```

Item 2a (5%) – Escreva uma função que mostre os LEDS de inicialização. A rotina deve acender os dois LEDs por um certo tempo, em seguida manter o LED verde aceso por um certo tempo, e finalmente manter os dois LEDS apagados por um certo tempo. A assinatura da função **deve** ser:

```
void start();
```

Item 2b (5%) – Escreva uma rotina para gerar uma sequência aleatória, utilizando os bits gerados pelo lfsr. O tamanho da sequência deve ser um parâmetro da função. A função deve funcionar para tamanhos entre 2 e 10. A assinatura da função **deve** ser (lembre-se de que a sequência é armazenada na variável global):

```
void createSequence(int n);
```

Caso o aluno não tenha resolvido a questão 1, ele pode utilizar a seguinte alternativa na solução das próximas questões (o funcionamento dessa versão alternativa é, naturalmente, diferente):

```
unsigned int mynumber = 0;
unsigned int lfsr()
{
   mynumber++;
   unsigned int x = (mynumber % 3);
   return (x % 2);
}
```

Item 2c (5%) – Escreva uma função para mostrar uma sequência aleatória gerada pela função do item 2b. Se o bit for 0, ela deve acender o LED vermelho por um tempo, e se o bit for 1 ela deve acender o LED verde pelo mesmo tempo. A assinatura da função deve ser (novamente, lembre-se que a sequência é armazenada em uma variável global):

```
void showSequence(int n);
```

Item 2d (25%) – Escreva uma rotina para receber a entrada do usuário e compará-la com a sequência desejada. O acionamento da chave deve ser completo, isto é, a chave deve ser **pressionada e solta** para que isso conte como um acionamento. Uma vez que uma das chaves foi acionada, não é necessário observar o comportamento da outra chave. A rotina deve retornar 1, caso as chaves sejam pressionadas na mesma ordem da sequência, ou 0, caso contrário. Em caso de uma entrada errada, não é necessário esperar que todas as teclas sejam pressionadas (isto é, caso a sequência esperada tenha tamanho 6, mas o primeiro acionamento já for incorreto, a rotina já pode retornar 0 instantaneamente, sem se preocupar com os demais caracteres. A assinatura da função **deve** ser:

```
int game(int n);
```

Item 2e (5%) – Escreva duas rotinas, uma para avisar que a sequência foi acionada corretamente (piscando o LED verde 5 vezes) e outra para avisar que a sequência foi acionada incorretamente (piscando o LED vermelho 5 vezes). A assinatura das funções **deve** ser:

```
void youwon(); //Sequência correta
void youlose(); //Sequência Incorreta
```

Item 2f (10%) – Complete o jogo incrementando o tamanho da sequência em caso de acerto, e reiniciando em caso de erro (recomeçando com tamanho 2). Caso o tamanho máximo (10) seja atingido, o programa deve travar piscando o LED verde indefinidamente.

QUESTÃO 3 – Interrupções (30%)

Em muitas aplicações é importante descobrir a ordem de acontecimento de dois eventos. Por exemplo, imagine que duas chaves vão ser acionadas em momentos próximos, e precisamos discernir qual das chaves foi acionada primeiro da forma mais precisa possível.

No nosso exemplo, escreva um programa para o MSP-430 que indique qual das duas chaves (P2.1 ou P1.1) foi acionada primeiro. Caso tenha sido a chave P2.1, o led vermelho (P1.0) deve acender, caso tenha sido a chave P1.1, o led verde (P4.7) deve acender. Quando as duas chaves forem soltas, o estado dos leds deve zerar e o programa deve recomeçar a monitorar as chaves novamente. Note que as duas chaves devem estar soltas ao mesmo tempo para que o programa reinicie. **Para receber pontos por esse programa é necessário utilizar interrupções nas duas chaves.**