Arquitetura de Computadores I
Trabalho retirado do Instituto De Informática - Departamento De Informática Aplicada UFRGs

Trabalho Prático - Simulador RAMSES

Escrever um programa para o simulador Ramses que faça a verificação de validade de dados contidos em um vetor, os quais estão codificados usando o código de Hamming de 7 bits descrito no Capítulo 8 (oito) do livro Fundamentos de Arquitetura de Computadores (Raul Weber).

O vetor estará armazenado na memória, a partir de um endereço inicial "**e**" e conterá "**n**" elementos. Cada elemento conterá um dado codificado, no qual os sete bits menos significativos (bits 0 a 6) serão o código e o bit mais significativo (bit 7) será sempre igual a zero.

Para cada elemento do vetor, o programa deverá ler o valor nele armazenado e verificar se o valor lido (excetuando o bit mais significativo) é um código válido. Se for um código válido, manter inalterado aquele elemento do vetor. Se não for um código válido, o programa deverá (1) corrigir o dado contido nos 7 bits menos significativos, de acordo com o algoritmo descrito no mesmo Capítulo 8 (oito) do livro texto, transformando-o em um código válido, e (2) colocar o bit mais significativo deste elemento em 1, para indicar que o mesmo foi alterado.

No final, o programa deverá indicar, numa variável "**errados**", quantos códigos foram corrigidos. O número de elementos do vetor e o endereço inicial do mesmo na memória estarão indicados, respectivamente, nas palavras 128 e 129. O total de elementos que estavam errados e foram corrigidos deve ser indicado na palavra 130:

128 - número de elementos do vetor (**n**)
129 - endereço inicial do vetor (**e**)
130 – total de elementos corrigidos (**errados**)

O programa não precisa se preocupar com a validação dos valores de "**n**" e "**e**", ou seja, deve supor que eles são coerentes e que estão de acordo com os seguintes limites:

$$1 \leq \mathbf{n} \leq 36$$
$$200 \leq \mathbf{e} + \mathbf{n} - \mathbf{1} \leq 255$$

Como os trabalhos serão corrigidos de forma automática, 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 de "**n**", "**e**" e "**errados**" devem ser exatamente os especificados acima
- o programa não deve alterar os valores das palavras 128 e 129
- o programa não deve assumir que o endereço 130 inicialmente contém zero
- não devem ser utilizadas pelo programa as palavras 200 a 255 da memória, reservadas para o vetor

O número de acessos à memória para processar cada caso de teste será medido

O programa, juntamente com uma listagem do programa em linguagem simbólica para o Daedalus, devidamente documentada, deverá ser entregue por e-mail (o assunto deve ser **[ARQ2006-HAMMING]** seguido pelo seu nome).

Exemplos de casos de teste (entrada), em binário:

n (pal. 128)	*e* (pal. 129)	*mem(e)*	*mem(e+1)*	*mem(e+2)*	*mem(e+3)*	*mem(e+4)*	*mem(e+5)*	*mem(e+6)*
1	240	00001110	X	X	X	X	X	X
3	200	00000000	01111111	01110000	X	X	X	X
2	254	01111000	01101001	X	X	X	X	X
7	220	01101010	00100000	01100001	01000111	01001110	00100100	00100110
6	210	01100000	01101001	01011010	01100110	00011001	01100111	X

Mesmos casos de teste (entrada), em decimal:

n (pal. 128)	*e* (pal. 129)	*mem(e)*	*mem(e+1)*	*mem(e+2)*	*mem(e+3)*	*mem(e+4)*	*mem(e+5)*	*mem(e+6)*
1	240	14	X	X	X	X	X	X
3	200	0	127	112	X	X	X	X
2	254	120	105	X	X	X	X	X
7	220	106	32	97	71	78	36	38
6	210	96	105	90	102	25	103	X

Resultados esperados dos casos de teste (saída), em binário:

n (pal. 128)	*e* (pal. 129)	*errados* (pal.130)	*mem(e)*	*Mem(e+1)*	*mem(e+2)*	*mem(e+3)*	*mem(e+4)*	*mem(e+5)*	*mem(e+6)*
1	240	1	10001111	X	X	X	X	X	X
3	200	0	00000000	01111111	01110000	X	X	X	X
2	254	1	11110000	01101001	X	X	X	X	X
7	220	7	10101010	10000000	11101001	11001100	11001100	10100101	11100110
6	210	2	11110000	001101001	01011010	01100110	00011001	11100110	X

Mesmos resultados esperados dos casos de teste (saída), em decimal:

n (pal. 128)	*e* (pal. 129)	*errados* (pal.130)	*mem(e)*	*mem(e+1)*	*mem(e+2)*	*mem(e+3)*	*mem(e+4)*	*mem(e+5)*	*mem(e+6)*
1	240	1	143	X	X	X	X	X	X
3	200	0	0	127	112	X	X	X	X
2	254	1	240	105	X	X	X	X	X
7	220	7	170	128	233	195	204	165	230
6	210	2	240	105	90	102	25	230	X

Data de Entrega: 13/06/2006

Texto extraído do capítulo 8 do livro “Fundamentos de Arquitetura de Computadores”, terceira edição:

O código de Hamming corrige um erro alterando a palavra errada para o código válido mais próximo (de menor distância), desde que esta menor distância seja inequívoca, isto é, não existam duas ou mais palavras corretas a igual distância da palavra errada.

Para possibilitar a correção de um erro simples, o código de Hamming mostrado na Tabela 8.10 a seguir utiliza três bits adicionais de paridade (A, B e C) para os quatro bits de código (ponderados por 8, 4, 2 e 1).

Posição	1	2	3	4	5	6	7
Código	A	B	8	C	4	2	1
0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1
2	0	1	0	1	0	1	0
3	1	0	0	0	0	1	1
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	1
6	1	1	0	0	1	1	0
7	0	0	0	1	1	1	1
8	1	1	1	0	0	0	0
9	0	0	1	1	0	0	1
10	1	0	1	1	0	1	0
11	0	1	1	0	0	1	1
12	0	1	1	1	1	0	0
13	1	0	1	0	1	0	1
14	0	0	1	0	1	1	0
15	1	1	1	1	1	1	1

Tabela 8.10 - Código de Hamming

Neste caso, o bit ‘A’ é usado para calcular a paridade par das posições 1, 3, 5 e 7. O bit ‘B’ é usado para obter-se paridade par nas posições 2, 3, 6 e 7 enquanto ‘C’ realiza a paridade par para as posições 4, 5, 6 e 7.

Existindo um erro, a posição do bit em erro é calculada por ‘cba’, onde ‘c’, ‘b’ e ‘a’ serão zeros se a paridade calculada por ‘C’, ‘B’ e ‘A’ for par ou “uns” se a paridade for ímpar. Por exemplo, se o número seis (1100110) for recebido erroneamente como 1100010, tem-se:

- verificação das posições 4, 5, 6 e 7: paridade ímpar: c = 1
- verificação das posições 2, 3, 6 e 7: paridade par: b = 0
- verificação das posições 1, 3, 5 e 7: paridade ímpar: a = 1

Assim, ‘cba’ = 101, ou seja, a posição 5 está errada e seu bit deve ser invertido para ser corrigido. Se o número tivesse sido recebido correto, a combinação dos bits de paridade seria zero (000).