Fundamentos da Computação Digital - MAB111
Nelson Quilula Vasconcelos - lula@im.ufrj.br
Respostas da segunda avaliação - 2020/2

1ª Questão (2 pontos)

Nos computadores eletrônicos modernos qualquer tipo de informação precisa ser sempre representada por conjuntos de bits. Quando as informações são apenas números, as escolhas óbvias são usar a representação binária desses números, ou a reprentação BCD ou excesso 3. Outros tipos de informação precisam ser codificadas de alguma outra forma.

Um tipo de informação que precisa frequentemente ser manipulada em computadores são textos. Como textos são formados por sequências de letras, espaços em branco, sinais de pontuação, mudanças de linhas, tabulações etc. É necessário usar um código que permita representar cada um desses elementos como padrões de bits. No passado foram usados diversos tipos de códigos mas, eventualmente, o código ASCII tornou-se um padrão.

O código ASCII foi definido no início da década de 60 do século vinte e previa inicialmante o uso de 3 conjuntos de 32 símbolos: no primeiro conjunto foram definidos 32 símbolos de controle; no segundo foram incuídos o espaço em branco, 21 sinais de pontuação e os dez algarismos decimais; no terceiro conjunto foram definidos códigos para as 26 letras maiúsculas em uso nos EUA e mais seis sinais de pontuação. Ainda na década de 60 do século vinte foi acrescentado um quarto grupo, formado por mais 32 símbolos que incluiam as 26 letras minúsculas usadoas em inglès e mais seis sinais de pontuação. Dessa forma, o código passou a empregar um total de 128 símbolos, representados pelos números decimais de 0 a 127 ou por números binários de sete bits.

Textos em inglês podem ser representados de forma muito conveniente usando o código ASCII. Nos casos de outras línguas faladas na europa ocidental e nas américas surge o problema de como representar certas letras que são usadas nessas outras línguas mas que não existem no inglês. Embora existam formas de contornar o problema, a solução natural foi acrescentar mais símbolos ao código ASCII para contemplar essas outras letras. Para permitir essa expansão, um oitavo bit foi acrescentado, possibilitando a inclusão de até mais 128 símbolos. Um problema é que essa extensão do código ASCII não foi realizada de uma forma coordenada, o que resultou em uma proliferação de inúmeras extensões diferentes. Eventualmente, a extensão definida pela “microsoft”, impropriamente denominada de ANSI, tornou-se dominante.

Mas alguns problemas subsistiam: Em diversos locais são usados alfabetos muito diferentes dos empregados na europa ocidental e nas américas, por exemplo o alfabeto usado na língua árabe, o alfabeto cirílico, o alfabeto usado na língua hebraica etc. A solução para acomodar os símbolos usados em todos esses alfabetos foi criar o padrão UNICODE que acrescenta mais oito bits aos códigos, possibilitando a codificação de 65536 diferentes símbolos, o que permite acomofar os símbolos usados em todos esses alfabetos. Mas essa solução faz com que cada letra de um texto ocupe o dobro do espaço porque poque passam a ser usados 16 bits para cada letra. Em textos escritos nas línguas usadas na europa ocidental isso parece um desperdício, porque a maior parte das letras desses textos pode ser representada usando o ASCII.

Para mitigar esse problema, foi criado o padrão UTF-8. Como funciona o processo de codificação usado no UTF-8?

Resposta:

A unidade básica de armazenamento usada no UTF-8 é um byte (8 bits) mas o código de um caracter pode ocupar de um a quatro bytes.

Caracteres que façam parte do conjunto ASCII são armazenados em apenas um byte. Como o ASCII usa códigos de apenas 7 bits, o bit mais significativo do byte será sempre zero.

Nos caracteres que não fazem parte do conjunto ASCII, o bit mais significativo de todos os bytes empregados é sempre um.

No primeiro byte de de cada código os dois bits mais significativos tem sempre o valor 11.

Nos demais bytes do código o valor dos dois bits mais siginificativos é sempre 10 e os seis demais bits informam o valor de um grupo de seis bits do código UNICODE do caracter.

O valor do primeiro byte dos códigos que ocupam mais de um byte também informa o número total de bytes empregados:

- Em códigos que ocupam dois bytes, os três bits mais significativos do primeiro byte tem o valor 110 e os cinco demais bits informam os cinco bits mais significativos do código UNICODE do caracter.;
- Em códigos que ocupam três bytes, os quatro bits mais significativos do primeiro byte tem o valor 1110 e os quatro demais bits informam os quatro bits mais significativos do código UNICODE do caracter;
- Em códigos que ocupam quatro bytes, os cinco bits mais significativos do primeiro byte tem o valor 11110 e os três demais bits informam os três bits mais significativos do código UNICODE do caracter;

Os códigos que ocupam dois bytes são formados pelos cinco bits informados no primeiro byte e por mais um grupo de seis bits informados no segundo byte, totalizando onze bits, o que permite referenciar os códigos UNICODE com valores menores que 2048. Esses códigos cobrem quase todos os caracteres usados nas línguas que usam o alfabeto latino e também os alfabetos grego, cirílico, copta, armenio, hebraico, árabe, siriaico, Thaana e N´Ko.

Os códigos que ocupam três bytes são formados pelos quatro bits informados no primeiro byte e por mais dois grupos de seis bits informados no segundo e terceiro bytes, totalizando dezesseis bits, o que permite referenciar os códigos UNICODE com valores menores 65536. Esses códigos abrangem toda a definição original do UNICODE (de 16 bits) que permite codificar quase todos os carateres atualmente em uso, incluindo a maior parte dos caracteres usados no chinês, japonês e coreano.

Os códigos que ocupam quatro bytes são formados pelos três bits informados no primeiro byte e por mais três grupos de seis bits informados no segundo, terceiro e quarto bytes, totalizando vinte e um bits, o que permite referenciar os códigos UNICODE com valores menores 2097152. Esses códigos abragem todo o UNICODE atualmente definido, incluindo os caracteres incomuns do chinês, japonês e coreano, diversas línguas históricas, símbolos matemáticos, emoji etc.

A tabela abaixo resume a codificação usada no UTF-8:

Tamanho	Byte 1	Byte 2	Byte 3	Byte 4	UNICODE	Último UNICODE	observações
1	0vvv vvvv				vvv vvvv	0x7F=127	ASCII
2	110v vvvv	10xx xxxx			vvv vvxx xxxx	0x7FF= 2047	Línguas alfabéticas
3	1110 vvvv	10xx xxxx	10yy yyyy		vvvv xxxx xxyy yyyy	0xFFFF= 65535	Quase todos os caracteres em uso
4	1111 0vvv	10xx xxxx	10yy yyyy	10zz zzzz	v vvxx xxxx yyyy yyzz zzzz	0x1FFFF= 2097151	Todos os caracteres

2ª Questão (1 pontos)

Como é possível representar sons em um computador?

Sons são ondas de pressão no ar.

Microfones são dispositivos que produzem uma saída elétrica que é proporcional a pressão em cada instante.

É possível converter o valor da saída elétrica produzida por um microfone para números binários empregando um conversor analógico digital.

O número de bits empregados depende da faixa de intensidades de sons que se pretende alcançar: Instensidades de sons são medidas usanda uma escala logarítmica cuja unidade é o Bell (1Bell = 10dB). Cada Bell corresponde a uma multiplicação da potência do som por dez. A faixa de intensidades de sons que o ouvido humano é capaz de usar é de cerca de 11 Bell.

A saída produzida pelos microfones é proporcional à raiz quadrada da potência naquele instante Para comportar toda a faixa de potência que nós conseguimos captar seria necessário empregar números na faixa de 0 até a raiz quadrada de 10^11= 10^5,5 ~= 316228. Esse é um número de 19 bits.

Na prática nenhum equipamento de reprodução de sons consegue alcaçar toda essa faixa de intensidadesde som. Por essa razão geralmente são empregrados, no máximo, números de 16 bits com sinal, O que restringe a faixa de intensidades de sons a 2 x log (32768) ~= 9 Bell. Em situações em que não é necessário usar uma faixa de potências tão ampla, esse número de bits pode ser reduzido. Usando números de dez bits com sinal, a faixa de potências é 2 x log (512) ~= 5,4 Bell, o que já é suficiente para reproduzir apropriadamente os sons da fala usual.

Além disso, para reproduzir corretamente um sinal, o limite de Nyquist informa que é necessário usar uma taxa de amostragem desse sinal pelo menos duas vezes maior que o componente de maior frequência presente no sinal. Em pessoas jovens, o limite de frequências percebidas pelo ouvido pode atingir 20KHz. Dessa forma, para reproduzir corretamente todos os sons que uma pessoa jovem é capaz de perceber é necessário usar uma taxa de amostragem superior a 40000 amostras por segundo. Uma taxa usualmente empregada é 44100 amostras por segundo.

Mas, para reconhecer corretamente a fala é suficiente preservar apenas as componentes com frequência até 3kHz, o que requer apenas uma taxa de amostragem maior que 6000 amostras por segundo.

3ª Questão (1 ponto)

Se o barramento de um computador usasse 20 bits para endereços e 16 bits para dados, qual seria a capacidade máxima de memória utilizável nesse computador?

2^20 * 16 bits = 2^24 bits = 2^21 bytes = 2Mbytes.

4ª Questão (1 ponto)

Descreva o ciclo de execução de uma instrução em computadores que usam o modelo de programa armazenado na memória (modelo de von Neumann).

Antes de uma instrução poder começar a ser executada é preciso ler o código da instrução da memória, ou seja, ler o conteúdo da posição de memória cujo endereço está armazenado no registrador denominado contador de programa ou apontador de instrução.

Antes, durante ou depois da execução das operações especificadas pela instrução o conteúdo desse registrador precisa ser alterado para que ele passe a armazenar o endereço de memória de onde será lido o código da instração a ser executada após o final da instrução corrente.

Em instruções que não vão gerar desvios no fluxo de controle, a alteração no valor armazenado nesse registrador é uma simples incrementação. Esse esquema faz que o código da próxima instrução a ser executada venha a ser lido da posição de memória cujo endereço é imediatamente seguinte àquele onde está armazenada a instrução que está sendo executada.

5ª Questão (1 ponto)

O que é e para que serve o teorema de De Morgan?

O teorema de De Morgan afirma que x & y = (x\ | y\ ) \ ou que x | y = (x\ & y\ )\.

Uma formulação alternativa é (x & y)\ = x\ | y\ ou também (x | y)\ = x\ & y\.

Esse teorema é útil na simplificação de expressões booleanas.

Em certas tecnologias é mais conveniente implementar operadores “não e” ou “não ou”. Nesse caso o teorema de De Morgan pode ser usado para converter algo como (x & y) | (x & z) para ((x & y)\ & (x & z)\ )\ que pode ser mais conveniente de implementar.

6ª Questão (1 ponto)

Apresente a implemelentação de “x ou exclusivo y” usando apenas as operações “não e” e “não”.

O conjunto um do ou exclusivo é { 01, 10 }.

Desta forma, OuExc (x, y) = (x\ & y) | (x & y\ ).

Usando De Morgan no | obtemos OuExc (x, y) = ((x\ & y)\ & (x & y\ )\ )\. Nessa expressão são usados apenas operações “não e” e não.

Usando operadores prexifados, podemos também escrever:

OuExc (x, y) = NãoE (NãoE (Não (x), y), NãoE (x, Não (y))).

7ª Questão (1 ponto)

Considere a função votador majoritário de cinco entradas u, v, x, y e z. Essa função produz como saída o valor lógico 1 se 3 ou mais de suas entradas tiverem o valor lógico 1. Apresente uma expressão booleana para essa função.

Resposta:

Uma abordagem direta é contsruir um ou “ou lógico” de todas as combinações de valores de cinco variáveis booleanas u, v, x, y e z em que três dessas cinco variáveis tem valor verdadeiro e o valor das outras duas variáveis é irrelevante:

```
( u & v & x ) | ( u & v & y ) | ( u & v & z ) |
( u & x & y ) | ( u & x & z ) | ( u & y & z ) |
( v & x & y ) | ( v & x & z ) | ( v & y & z ) | ( x & y & z )
```

Outra abordagem é enumerar os números binários de cinco bits em que três ou mais bits tem valor um para obter os 16 elementos do conjunto um da função votador majoritário de cinco entradas:: 00111, 01011, 01101, 01110, 01111, 10011, 10101, 10110, 10111, 11001, 11010, 11011, 11100, 11101, 11110 e 11111.

Assim uma expressão lógica para essa função pode ser construída fazendo um “ou lógico`das 16 sub-expressões correspondentes a cada um dos 16 elementos do conjunto um, obtendo: ((u\ & v\ & x & y & z) | (u\ & v & x\ & y & z) | (u\ & v & x & y\ & z) | …

Uma terceira abordagem é construir um mapa de Karnaugh com dois planos a partir do conjunto um da função. Essa abordagem é um pouco mais trabalhosa:

Plano u = 0				vx	Plano u = 1			
yz=00	yz=01	yz=11	yz=10		yz=00	yz=01	yz=11	yz=10
0	0	0	0	0 0	0	0	1	0
0	0	1	0	0 1	0	1	**1**	1
0	1	**1**	1	1 1	1	**1**	**1**	**1**
0	0	1	0	1 0	0	1	**1**	1

É possível constatar que todos os cinco uns do plano 0 tem vizinhos com valor um no pano um. Portanto todos os quatro grupos formados no plano 0 são biplanares. O um que está em negrito no plano 0 faz parte de todos os quatro grupos formados nesse plano. Esses quatro grupos são: x & y & z, v & y & z, v & x & y e v & x & z.

Os cinco uns do segundo plano em negrito fazem parte desses grupos biplanares. Para contemplar os seis demais uns do segundo plando é necessário formar mais seis grupos de quatro uns: um horizontal na linha vx=11, um vertical na coluna yz=11 e quatro quadrados: x & y, v & y, x & z e v & z.

Explicitando os dez grupos, obtemos uma expressão equivalente à obtida na primeira abordagem.

8ª Questão (2 pontos)

Uma forma de apresentar os valores de algarismos decimais utiliza 5 lâmpadas ou leds que são dispostos nos vértices de um pentágono regular. Um desses leds, geralmente o led que

ocupa o vétrice superior do pentágono, é rotulado com o valor 0. Os outros quatro leds são rotulados 2, 4, 6 e 8, percorrendo o pentágono no sentido horário. Isso forma um padrão de leds semelhante ao seguinte:

0

8 **2**

6 **4**

Cada um dos dez possíveis valores de algarismos decimais são apresentados acendendo um ou dois leds, de acodo com a seguinte tabela:

Algarismo:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Leds acesos:	**0**	**0, 2**	**2**	**2, 4**	**4**	**4, 6**	**6**	**6, 8**	**8**	**8, 0**

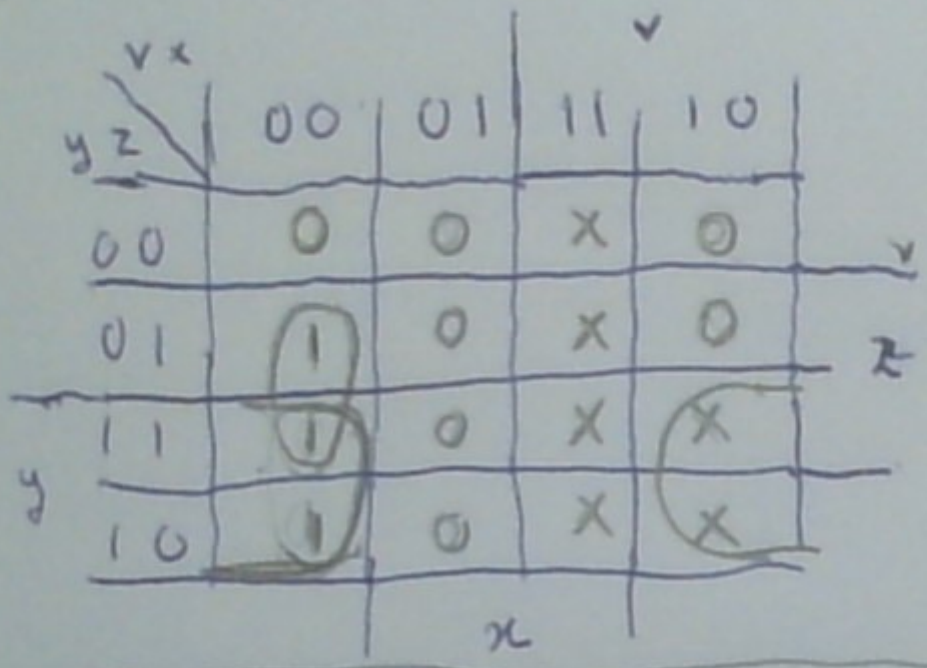
Observe que esse esquema produz uma forma de apresentação intuitiva, em que os algarismos com valor par são apresentados acendendo o led rotulado com o valor correspondente e os algarismos com valor ímpar são apresentados acendendo simultaneamente dois leds que correspondem aos valores imediatamente anterior e imediatamente seguinte ao valor do algarismo.

Considere que temos valores BCD, cujos quatro bits são representados pelas letras v, x, y e z, cujos valores queremos apresentar usando esse esquema de 5 leds. Para isso podemos construir a seguinte tabela verdade:

Algarismo:		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Valor BCD:	**v**	0	0	0	0	0	0	0	0	1	1
	x	0	0	0	0	1	1	1	1	0	0
	y	0	0	1	1	0	0	1	1	0	0
	z	0	1	0	1	0	1	0	1	0	1
Leds acesos:	**0**	**1**	**1**	0	0	0	0	0	0	0	**1**
	2	0	**1**	**1**	**1**	0	0	0	0	0	0
	4	0	0	0	**1**	**1**	**1**	0	0	0	0
	6	0	0	0	0	0	**1**	**1**	**1**	0	0
	8	0	0	0	0	0	0	0	**1**	**1**	**1**

Empregando mapas de Karnaugh, obtenha expressões do tipo “AND-OR” (“soma de produtos”) para as funções de v, x, y e z correspondentes aos leds 2 e 8.

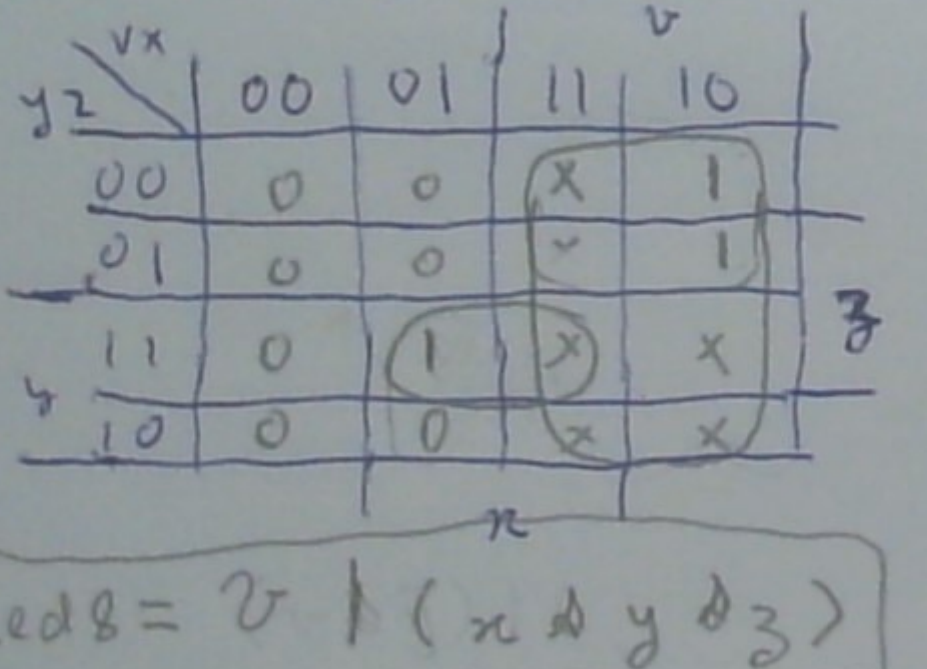
LED 2:



yz \ vx	00	01	11	10
00	0	0	x	0
01	1	0	x	0
11	1	0	x	x
10	1	0	x	x

$$Led2 = (\bar{v} \cdot \bar{x} \cdot z) + (\bar{x} \cdot y)$$

LED 8:



yz \ vx	00	01	11	10
00	0	0	x	1
01	0	0	x	1
11	0	1	x	x
10	0	0	x	x

$$Led8 = v + (x \cdot y \cdot z)$$