Arquitetura e Organização de Computadores

Exercícios

António José Araújo

João Canas Ferreira

Bruno Lima

Mestrado Integrado em Engenharia Informática e Computação

Outubro de 2020

Conteúdo

1 Aritmética binária **1**
- 1.1 Exercícios resolvidos 1
- 1.2 Exercícios propostos 7

2 Vírgula flutuante **11**
- 2.1 Exercícios resolvidos 11
- 2.2 Exercícios propostos 17

3 Circuitos combinatórios **20**
- 3.1 Exercícios resolvidos 20
- 3.2 Exercícios propostos 29

4 Circuitos sequenciais **35**
- 4.1 Exercícios resolvidos 35
- 4.2 Exercícios propostos 42

5 Desempenho **47**
- 5.1 Exercícios resolvidos 47
- 5.2 Exercícios propostos 52

6 Linguagem *assembly* **56**
- 6.1 Exercícios resolvidos 56
- 6.2 Exercícios propostos 62

7 Organização de um CPU **66**
- 7.1 Exercícios resolvidos 66
- 7.2 Exercícios propostos 72

8 Memória Cache **75**
- 8.1 Exercícios resolvidos 75
- 8.2 Exercícios propostos 77

Soluções dos exercícios propostos **81**
- 1 Aritmética binária 81
- 2 Vírgula flutuante 83
- 3 Circuitos combinatórios 85
- 4 Circuitos sequenciais 92
- 5 Linguagem *assembly* 94
- 6 Organização de um CPU 100
- 7 Memória Cache 101
- 8 Desempenho 103

Esta coletânea reúne exercícios resolvidos e propostos sobre a matéria lecionada em 2020/21 na unidade curricular de *Arquitetura e Organização de Computadores* do 1º ano do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto.

1 Aritmética binária

1.1 Exercícios resolvidos

Exercício 1
Realizar as conversões de base indicadas.

a) $72_{10} = ?_2 = ?_{16}$ b) $259_{10} = ?_2 = ?_{16}$ c) $1110_2 = ?_{10} = ?_{16}$

d) $100000{,}11_2 = ?_{10} = ?_{16}$ e) $1\text{BEEF}_{16} = ?_2$

a) A conversão de decimal para qualquer base de representação pode ser feita por divisões sucessivas pela base pretendida ou realizando a decomposição do número em potências dessa base. Optando por este segundo processo, resulta

$$\begin{aligned} 72_{10} = 64 + 8 = 2^6 + 2^3 &= 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \\ &= 1001000_2 \end{aligned}$$

Relativamente à conversão para hexadecimal, a forma mais simples de a realizar consiste em considerar a representação binária e formar grupos de 4 bits, da direita para a esquerda, e depois fazer a correspondência entre cada um desses grupos e o respetivo símbolo hexadecimal.

$$\underbrace{100}_{4}\ \underbrace{1000}_{8} = 48_{\text{H}}$$

Nota: O índice 'H', tal como '16', indica uma representação em hexadecimal.

b)

$$259_{10} = 256 + 2 + 1 = 2^8 + 2^1 + 2^0 = 100000011_2$$

$$100000011_2 = \underbrace{1}_{1}\ \underbrace{0000}_{0}\ \underbrace{0011}_{3} = 103_{\text{H}}$$

c)

$$1110_2 = 2^3 + 2^2 + 2^1 = 14_{10}$$

$$\underbrace{1110}_{\text{E}} = \text{E}_{\text{H}}$$

d)

$$100000{,}11_2 = 2^5 + 2^{-1} + 2^{-2} = 32{,}75_{10}$$

Como se trata de um número fracionário, na conversão de binário para hexadecimal, os grupos de 4 bits formam-se a partir da vírgula.

$$100000{,}11_2 = \underbrace{10}_{2}\ \underbrace{0000}_{0},\ \underbrace{1100}_{\mathrm{C}} = 20{,}\mathrm{C_H}$$

e)

$$1\mathrm{BEEF_H} = \underbrace{1}_{0001}\ \underbrace{B}_{1011}\ \underbrace{E}_{1110}\ \underbrace{E}_{1110}\ \underbrace{F}_{1111} = 11011111011101111_2$$

Exercício 2

Efetue as seguintes operações aritméticas binárias, considerando os operandos representados como números sem sinal, isto é, números positivos.

a) $101110 + 100101$

b) $1110010 - 1101101$

c) 1001011×11001

d) $1101000 \div 100$

As regras de cálculo são idênticas às regras usadas em base 10.

a)

$$\begin{array}{r} 101110 \\ +\underline{100101} \\ 1010011 \end{array}$$

b)

$$\begin{array}{r} 01\ 0 \\ 11\not{0}0\not{1}0 \\ -\underline{1101101} \\ 101 \end{array}$$

c)

$$\begin{array}{r} 1001011 \\ \times\ \ \underline{11001} \\ 1001011 \\ 1001011 \\ +\underline{1001011\ \ \ } \\ 11101010011 \end{array}$$

d) Nesta divisão ocorre uma situação particular: o divisor é uma potência de 2 ($100_2 = 2^2 = 4$). Nestas circunstâncias, o quociente pode ser obtido deslocando os bits do dividendo n posições para a direita, o que corresponde a subtrair n ao expoente de cada potência de base 2 da decomposição do dividendo. Neste exercício $n = 2$, pelo que, $1101000 \div 100 = 11010_2$.

Exercício 3

Represente os seguintes números decimais em sinal e grandeza e em complemento para 2 com 6 bits.

a) $+12$ b) -12 c) -1 d) $+32$

a) $12 = 8 + 4 = 1100_2$

- Sinal e grandeza: sendo o número positivo, o bit de sinal é 0; a grandeza, escrita em 5 bits, é 01100. Assim, $12 = 001100_2$.
- Complemento para 2: a representação de um número positivo em complemento para 2 é a mesma do número sem sinal (binário puro), respeitando porém a largura de representação. Portanto, $12 = 001100_2$.

b) A grandeza de -12 é $12 = 1100_2$.

- Sinal e grandeza: sendo o número negativo, o bit de sinal é 1; a grandeza é codificada com 5 bits, 01100. Assim, $-12 = 101100_2$.
- Complemento para 2:

  A representação de um número negativo em complemento para 2 pode ser obtida a partir da representação binária do simétrico do número. Para tal, copiam-se todos os bits da direita para a esquerda até encontrar o primeiro 1, que ainda é copiado, e a partir daí complementam-se os restantes bits. Embora existam outros processos, este é um processo expedito.

  Assim, partindo de $12 = 001100_2$ obtém-se $-12 = 110100_2$.

c)
- Sinal e grandeza: $-1 = 100001_2$.
- Complemento para 2: tendo em consideração o que foi descrito na alínea anterior, $1 = 000001_2$, pelo que $-1 = 111111_2$.

d) O número em causa é $32 = 2^5 = 100000_2$.

- Sinal e grandeza: a grandeza de 32 não se consegue codificar com apenas 5 bits, pelo que o valor 32 não é representável no formato pretendido.
- Complemento para 2: o valor 32 também não é representável com 6 bits; é um número positivo e no entanto 100000_2 representa um número negativo (MSB=1).

Com 6 bits, o maior número representável é $2^{6-1} - 1 = 31$.

Exercício 4

Considere a representação binária dos valores $C1_{16}$ e $A7_{16}$ com 8 bits.

a) Indique o valor decimal correspondente, admitindo que são interpretadas como números:

tskip(0) positivos; tskip(0) sinal e grandeza;

tskip(0) complemento para dois.

b) Indique a gama de valores representáveis considerando a forma complemento para 2.

c) Admitindo que a referida representação se encontra em complemento para dois, efetue a sua adição em binário e comente o resultado.

a) $C1_{16} = 11000001_2$ e $A7_{16} = 10100111_2$

i)

$$11000001_2 = 2^7 + 2^6 + 2^0 = 128 + 64 + 1 = 193$$

$$10100111_2 = 2^7 + 2^5 + 2^2 + 2^1 + 2^0 = 128 + 32 + 4 + 2 + 1 = 167$$

ii)

$$\underset{-}{\overset{1}{\sqcup}} \underbrace{1000001}_{2^6 + 2^0} = -65$$

$$\underset{-}{\overset{1}{\sqcup}} \underbrace{0100111}_{2^5 + 7} = -39$$

iii) Os valores são ambos negativos.

$$11000001_2 \overset{\text{compl. 2}}{\longrightarrow} 00111111_2 = 63$$

Este é o valor simétrico do número a identificar. Logo, 11000001_2 interpretado em complemento para 2, resulta no decimal -63.

$$10100111_2 \overset{\text{compl. 2}}{\longrightarrow} 01011001_2 = 89$$

Da mesma forma, 10100111_2 interpretado em complemento para 2, resulta no decimal -89.

b) Com 8 bits conseguem escrever-se $2^8 = 256$ números. Considerando que metade são números negativos e a outra metade corresponde a números positivos, incluindo o 0, resulta o seguinte intervalo de representação: $[-128; +127]$. Ao contrário do que sucede em sinal e grandeza, o 0 só tem uma representação.

De forma mais genérica, em complemento para 2, a gama de representação correspondente a n bits é:

$$[-2^{n-1}; +2^{n-1} - 1]$$

c) A soma faz como para números sem sinal, ignorando o transporte a partir do bit mais significativo.

$$\begin{array}{r} 11000001 \\ +\underline{10100111} \\ \not{1}01101000 \end{array}$$

O resultado encontrado (01101000_2) está incorreto, porque a adição de dois números negativos não pode resultar num número positivo. Ocorre portanto, *overflow*.

Pode confirmar-se esta conclusão em decimal: $(-63) + (-89) < -128$, isto é, a soma não é representável com 8 bits.

Exercício 5

Admitindo que $A = 11001_2$ e $B = 11101_2$ se encontram representados em complemento para 2 com 5 bits, calcule $A + B$ e indique, justificando, se ocorre *overflow*.

Efetuando os cálculos:

$$\begin{array}{r} 11001 \\ +\underline{11101} \\ \not{1}10110 \end{array}$$

Como se trata de uma adição em complemento para 2, o *carry* que ocorreu ao somar os bits mais significativos deve ignorar-se. A ocorrência deste *carry* não deve ser interpretada como ocorrência de *overflow*. Logo, $A + B = 10110$.

O resultado encontrado é válido no formato especificado, pois não ocorre *overflow*, porque da adição de dois números negativosresultou um número negativo.

Exercício 6

Considere dois números binários com 6 bits, $M = 101100_2$ e $N = 110010_2$. M está representado em sinal e grandeza e N está representado em complemento para 2.

a) Escreva M em formato hexadecimal e N em formato decimal.

b) Indique, justificando, qual dos números tem maior grandeza.

c) Mostre que em complemento para 2 a operação $M + N$ não produz *overflow*.

a) A conversão para hexadecimal é direta:

$$M = \underbrace{10}_{2}\ \underbrace{1100}_{\mathrm{C}} = 2\mathrm{C}_{\mathrm{H}}$$

Como N está em complemento para 2 e é negativo, $-N = 001110 = 14$, pelo que $N = -14$.

b) Como M está definido em sinal e grandeza e são usados 6 bits, a grandeza é dada pelos 5 bits menos significativos.

$$|M| = 01100 \qquad (= 12_{10})$$

Quanto a N, conclui-se da alínea anterior que

$$|N| = -N = 01110 \qquad (= 14_{10})$$

Conclusão: N é o número que possui maior grandeza.

c) O número M está definido em sinal e grandeza e é negativo, pelo que deve ser representado em complemento para 2 antes de ser operado.

$$-M = 001100 \quad \overset{\text{compl. 2}}{\longrightarrow} \quad M = 110100$$

N já se encontra em complemento para 2, pelo que se pode então calcular $M + N$:

$$\begin{array}{r} 110100 \\ +\underline{110010} \\ \not{1}100110 \end{array}$$

Não há *overflow*, pois os operandos são negativos e o resultado encontrado também é negativo.

Exercício 7

Os valores $X = 10111100_2$ e $Y = 10100110_2$ estão representados com 8 bits. Determine o menor número de bits necessário para representar corretamente $X + Y$, supondo que os operandos estão representados em:

a) sinal e grandeza;

b) complemento para 2.

a) Representação em sinal e grandeza:

$$X = \underset{S}{\underbrace{1}} \ \underbrace{0111100}_{G} \qquad \text{e} \qquad Y = \underset{S}{\underbrace{1}} \ \underbrace{0100110}_{G}$$

Para calcular $X + Y$ é necessário somar as grandezas de X e Y, operação esta realizada em 7 bits.

$$\begin{array}{r} 0111100 \\ +\underline{0100110} \\ 1100010 \end{array}$$

A grandeza resultante é representável em 7 bits. Como tal, e tendo em consideração o bit de sinal, o menor número de bits necessário para a representação de $X + Y$ em sinal e grandeza é 8.

b) Representação em complemento para 2:

$$\begin{array}{rr} \text{X+Y:} & 10111100 \\ & +\underline{10100110} \\ & \not{1}01100010 \end{array}$$

Estão a somar-se dois números com o mesmo sinal (negativos), mas o resultado alcançado com 8 bits apresenta sinal diferente (positivo). Quer isto dizer que ocorre *overflow* ao realizar a operação com 8 bits. Portanto, para que $X + Y$ seja corretamente representado em complemento para 2, o número mínimo de bits a considerar deverá ser 9.

Exercício 8
Considere a representação em complemento para 2 dos valores X e Y indicada:

$$X = 110010_2 \qquad \text{e} \qquad Y = 101011_2$$

a) Determine o valor decimal de X e Y.

b) Calcule $X - Y$ e $X + Y$. Comente os resultados.

a) X e Y são negativos, pelo que podem obter-se os números simétricos calculando o complemento para 2:

$$-X = 001110_2 = 2^3 + 2^2 + 2^1 = 14$$
$$-Y = 010101_2 = 2^4 + 2^2 + 2^0 = 21$$

Logo, $X = -14$ e $Y = -21$.

b) A diferença pode ser calculada pela adição do simétrico:

$$\begin{array}{rr} X - Y = X + (-Y): & 110010 \\ & +\underline{010101} \\ & \not{1}000111 \end{array}$$

Não ocorre *overflow*, porque os operandos têm sinais opostos. O resultado é por isso correto.

$$\begin{array}{rr} X + Y: & 110010 \\ & +\underline{101011} \\ & \not{1}011101 \end{array}$$

Ocorre *overflow*, porque os operandos têm o mesmo sinal e o resultado tem sinal oposto. Por este motivo, o resultado é errado, pois não é representável com os 6 bits considerados.

1.2 Exercícios propostos

Exercício 9
Em cada alínea, considere o número dado e represente-o nos sistemas de numeração indicados.

a) $256_{10} = ?_2 = ?_{16}$
b) $2047_{10} = ?_2 = ?_{16}$
c) $24{,}25_{10} = ?_2 = ?_{16}$
d) $4{,}2_{10} = ?_2 = ?_{16}$
e) $10000_2 = ?_{10} = ?_{16}$
f) $100{,}001_2 = ?_{10} = ?_{16}$
g) $1E_{16} = ?_2 = ?_{10}$
h) $ABCD_{16} = ?_{10} = ?_2$
i) $AB{,}C_{16} = ?_{10} = ?_2$
j) $1110_{10} = ?_{16}$

Exercício 10

Efetue as seguintes operações aritméticas binárias, considerando os operandos representados como números sem sinal, isto é, números positivos.

a) $110101 + 11001$
b) $101{,}01 + 100{,}111$
c) $101110 - 100101$
d) $1000010 - 101101$
e) $11011101 - 1100011$
f) $11011101 - 11000{,}11$
g) 1011×100

Exercício 11

Considere os números decimais $+3$, $+2$ e -3. Nas alíneas seguintes admita a representação em sinal e grandeza com 4 bits.

a) Escreva os números em binário.

b) Calcule $3 + 2$ e $2 + (-3)$.

c) Calcule $3 + 14$ e comente o resultado.

Exercício 12

Enumere os valores decimais que se podem representar com 4 bits usando as representações em sinal e grandeza e em complemento para 2. Em ambos os casos, indique a gama de representação na forma de um intervalo, relacionando os valores extremos (maior positivo e menor negativo) com o número de bits.

Exercício 13

Recorrendo a 8 bits, represente em sinal e grandeza e em complemento para 2, os seguintes números decimais:

a) 18
b) 49
c) -49
d) -3
e) -100
f) 115
g) -127
h) -128

Exercício 14

Considere os números $M = 33_{16}$ e $N = 33_{10}$ representados por 8 bits.

a) Calcule $M + N$ em binário, supondo que M e N são números sem sinal.

b) Admitindo que os valores estão representados em complemento para 2, diga se ocorre *overflow* ao calcular $N - M$.

Exercício 15

Admitindo que P e Q representam dois números binários em complemento para dois, com 8 bits, efetue a sua adição binária e interprete o resultado.

a) $P = \mathrm{DE_H}$ e $Q = \mathrm{A3_H}$
b) $P = \mathrm{8C_H}$ e $Q = \mathrm{D3_H}$
c) $P = \mathrm{8C_H}$ e $Q = 74_{\mathrm{H}}$

Exercício 16

Considere os seguintes números binários: $X = 11100011$ e $Y = 01001000$.

a) Indique o valor decimal de X e Y, para os casos de representação sem sinal e representação em complemento para 2.

b) Calcule $X + Y$ para ambas as representações de 8 bits e comente os resultados obtidos.

Exercício 17

Considere os números $P = 1111010_2$ e $Q = 0100010_2$ com 7 bits.

a) Escreva P em hexadecimal e Q em decimal.

b) Calcule $P + Q$ e comente o resultado considerando que P e Q representam números:

tsk[a] sem sinal; tsk[b] em complemento para 2.

Exercício 18

Considere os números $A = 11010_2$ (com 5 bits) e $B = 0101010_2$ (com 7 bits).

a) Considere que A e B representam números sem sinal. Calcular $A + B$ em 7 bits. Converter o resultado para decimal.

b) Repita a alínea anterior, considerando agora que A e B representam números representados em sinal e grandeza.

c) Repita a alínea anterior, considerando agora que A e B representam números representados em complemento para dois.

Exercício 19

Considere os números $M = 4\mathrm{A}_\mathrm{H}$ e $N = \mathrm{A4}_\mathrm{H}$ representados em complemento para 2 com 8 bits.

a) Escreva M em binário.

b) Determine o valor decimal de N.

c) Calcule $M - N$ e $N - M$, justificando se ocorre *overflow*.

Exercício 20

Considere os números $S = 11001000_2$ e $T = 00010001_2$ representados em complemento para 2 com 8 bits.

a) Determine o valor decimal de S e T.

b) Represente S e T em sinal e grandeza.

c) Calcule $S + T$ em sinal e grandeza e comente o resultado encontrado.

Exercício 21

Considere os números de 8 bits expressos, nas bases indicadas, por: $X_2 = 10111100$, $Y_{10} = 73$ e $Z_{16} = 9\text{E}$.

a) Escreva Y em hexadecimal e Z em binário.

b) Calcule $X + Z$, considerando que os operandos X e Z estão em complemento para 2, e justifique se ocorre *overflow*.

c) Calcule o maior número N, em complemento para 2 com 8 bits, que adicionado a X conduz a uma soma negativa.

Exercício 22

Suponha que se pretende calcular a seguinte expressão (em que x é um número inteiro):

$$y = x^2 - 30x + 161$$

Considere x representado em binário com 5 bits (sem sinal).

a) Qual é a gama de valores de x?

b) Determinar o maior valor positivo de y.

c) Determinar o valor mais negativo de y?

d) Qual é o número mínimo de bits necessário para representar o valor com sinal y.

e) Representar os valores determinados nas alíneas b) e c) em complemento para dois.

2 Vírgula flutuante

2.1 Exercícios resolvidos

Exercício 1
A representação dos números reais X e Y no formato de precisão simples da norma IEEE 754 é a seguinte:

X: C3800000_H $\qquad$ Y: $00111111100000000000000000000000_2$

a) Calcule o expoente real do número codificado em X.

b) Determine o sinal de $X + Y$.

c) Justifique a afirmação: Sendo X um número qualquer, $X \times Y = X$.

a)

$$X = \text{C3800000}_\text{H} = 1 \underbrace{10000111}_{E_X = 135} 00000000000000000000000_2$$

O expoente real de X é:

$$E_X^{\text{real}} = 135 - 127 = 8$$

b) Como $E_Y = 127$, $E_X > E_Y$. Então $|X| > |Y|$ e portanto,

$$S_{X+Y} = S_X = 1$$

ou seja, $X + Y$ é um número negativo.

c) Como $E_Y = 127$, $E_Y^{\text{real}} = 0$.
Porque os 23 bits da mantissa de Y são nulos, $M_Y = 1{,}0$.
Então, conclui-se que $Y = 1$ e por isso, para qualquer X, tem-se $X \times Y = X$.

Exercício 2
Considere $Y = 25{,}25_{10}$ e o número real X cuja representação em formato IEEE 754 (precisão simples) é BF400000_{16}.

a) Mostre a representação de Y no formato IEEE 754 (em binário).

b) Calcule $X \times Y$, indicando claramente todos os passos efetuados.

a) Pretende-se mostrar Y em binário no contexto da representação em vírgula flutuante com precisão simples (32 bits).

$$Y = 25{,}25_{10} = 2^4 + 2^3 + 2^0 + 2^{-2} = 11001{,}01_2 = 1{,}100101_2 \times 2^4$$

- $S_Y = 0$
- $E_Y = E_Y^{real} + 127 = 4 + 127 = 131 = 10000011_2$
- $M_Y = 1{,}100101_2$

Resulta então:

$$Y = 01000001110010100\underbrace{\cdots 0}_{17\ 0\text{'s}}$$

b)

$$X = \text{BF400000}_{16} = \underbrace{1}_{S_X}\ \underbrace{01111110}_{E_X}\ \underbrace{100\cdots0_2}_{f_X}$$

Então:

- $S_{X\times Y} = 1$
- $E_{X\times Y} = E_X + E_Y - 127 = (131 + 126) - 127 = 130 = 10000010_2$
- $M_{X\times Y}$: (em binário)

$$\begin{array}{r} 1{,}1001010\cdots\ 0 \\ \times 1{,}1000000\cdots\ 0 \\ \hline 0\ 0000000\cdots\ 0 \\ \cdots \\ 11001010\cdots\ 0 \\ 11001010\cdots\ 0 \\ \hline 10{,}01011110 \qquad \cdots \qquad 0 \end{array}$$

A necessidade de normalizar a mantissa resultante leva ao incremento do expoente calculado:

$$M_{X\times Y} = 10{,}01011110\cdots0_2 = 1{,}00101111_2 \times 2^1$$

Assim, $E_{X\times Y} = 131_{10} = 10000011_2$, pelo que

$$X \times Y = 11000001100101111\underbrace{00\cdots0}_{15\ 0\text{'s}}$$

Exercício 3

Considere o número cuja representação em hexadecimal é C1200000_H. Indique o valor decimal correspondente, se assumir que o número está representado:

a) como inteiro sem sinal;

b) em complemento para 2;

c) em vírgula flutuante com precisão simples.

a) A representação binária do número é:

$$\text{C1200000}_\text{H} = 1100\ 0001\ 0010\ 0000\ 0000\ 0000\ 0000\ 0000_2$$

Interpretando este número como um inteiro sem sinal, o valor decimal correspondente é determinado com base na posição que cada bit ocupa. Assim:

$$\begin{aligned} 11000001001000000000000000000000_2 &= 2^{31} + 2^{30} + 2^{24} + 2^{21} \\ &= 2^{21} \times (2^{10} + 2^9 + 2^3 + 2^0) \\ &= 2 \times 2^{10} \times 2^{10} \times (1024 + 512 + 9) \\ &= 2097152 \times 1545 \\ &= 3240099840_{10} \end{aligned}$$

Nota: É admissível apresentar a solução na forma de uma soma de potências de 2.

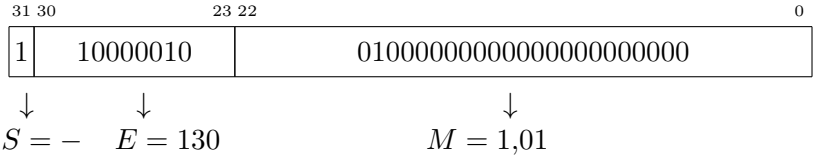
b) $\text{C1200000}_\text{H} = 1100\ 0001\ 0010\ 0000\ 0000\ 0000\ 0000\ 0000_2$

Interpretando este número dado em complemento para 2 e atendendo a que é um número negativo (MSB=1), o decimal correspondente pode ser determinado depois de obter o simétrico correspondente. Este pode ser obtido por aplicação da regra prática que consiste em copiar todos os bits da direita para a esquerda até ser encontrado o primeiro 1 e depois complementar os restantes. Assim:

$$\begin{aligned} &11000001001000000000000000000000_2 \\ &\downarrow \qquad\qquad \downarrow \\ &-01111101110000000000000000000000_2 \\ \\ &= 2^{29} + 2^{28} + 2^{27} + 2^{26} + 2^{25} + 2^{23} + 2^{22} + 2^{21} \\ &= 1054867456_{10} \end{aligned}$$

Portanto, C1200000_H representa -1054867456_{10} em complemento para 2.

c) Assumindo o formato de vírgula flutuante, a interpretação do padrão de bits é:



Como o expoente está representado em excesso 127, o expoente real é 3. Então, o número decimal correspondente será:

$$-1{,}01_2 \times 2^3 = -1010_2 = -10_{10}$$

Exercício 4

Sejam X e Y dois números reais, representados em vírgula flutuante com o formato de precisão simples da norma IEEE 754 da seguinte forma:

$$X\text{: } 11000011000000110000000000000000_2$$
$$Y\text{: } 11000011011111000000000000000000_2$$

a) Mostre que X é um número inteiro.

b) Calcule o número Z que verifica a condição $X + Z = 0$.

c) Mostre que o expoente real de $X + Y$ é 8.

a)

$$X = \underset{-}{\underbrace{1}} \quad \underbrace{10000110}_{E_X = 134} \quad \underbrace{00000110000000000000000_2}_{M_X = 1{,}0000011}$$

O expoente real de X é:

$$E_X^{\text{real}} = 134 - 127 = 7$$

Logo,

$$\text{X}_{10} = -1{,}0000011_2 \times 2^7 = -10000011_2 = -131_{10}$$

b)

$$\text{X} + \text{Z} = 0 \quad \Leftrightarrow \quad \text{Z} = -\text{X}$$

Em termos de representação em vírgula flutuante

$$\begin{aligned} S_Z &= -S_X \\ E_Z &= E_X \\ M_Z &= M_X \end{aligned}$$

Logo,

$$\text{Z} = 01000011000000110\underbrace{0\,0\cdots0_2}_{16\ 0\text{'s}}$$

c)

$$E_{X+Y}^{\text{real}} = E_Y^{\text{real}} \equiv E_X^{\text{real}} = 7$$

Contudo, ao somar as mantissas, o resultado pode não estar normalizado:

$$\begin{array}{lrcl} M_X: & 1{,}0000011 & & \\ M_Y: & \underline{+1{,}1111100} & & \\ M_{X+Y} & 10{,}1111111 & = & 1{,}01111111_2 \times 2^1 \end{array}$$

Logo, o expoente indicado inicialmente tem de ser incrementado, resultando

$$E_{X+Y}^{\text{real}} = 8$$

Exercício 5

Os números reais X, Y e Z estão representados no formato de precisão simples da norma IEEE 754.

a) Sendo X representado por 23456765432_8, indique o seu sinal.

b) Complete, com os bits em falta, os campos da seguinte igualdade:

$$Y = +\underline{\qquad\qquad} \times 2^3 = \underbrace{}_{1\text{ bit}}\ \underbrace{\qquad\qquad\qquad}_{8\text{ bits}}\ \underbrace{001010\quad ...\quad 0}_{23\text{ bits}}$$

c) Qual é o número Z tal que o valor de $Y \times Z$ é representado por $110000011001010\cdots0_2$?

a)

$$X = 23456765432_8 = \underbrace{1}_{\substack{\downarrow\\ -}}\ \underbrace{00111001}_{E_X}\ \underbrace{01110111110101100011010}_{f_X}$$

X é negativo.

Note-se que a conversão de cada dígito octal origina 3 bits e por essa razão X_2 devia possuir 33 bits. Porém, a representação de X_2 em vírgula flutuante é constituída por 32 bits, que no seu conjunto correspondem a X_8, pois o dígito mais significativo (2) pode escrever-se como 010_2 ou 10_2.

b)

$$\begin{array}{lcl} Y > 0 & \Rightarrow & S_Y = 0 \\ E_Y^{\text{real}} = 3 & \Rightarrow & E_Y = E_Y^{\text{real}} + 127 = 130 = 10000010_2 \\ f_Y = 0{,}00101 & \Rightarrow & M_Y = 1{,}00101_2 \end{array}$$

Daqui resulta

$$Y = +1{,}00101_2 \times 2^3 = \underbrace{0}_{1\text{ bit}}\ \underbrace{10000010}_{8\text{ bits}}\ \underbrace{001010...0}_{23\text{ bits}}$$

c)

$$\begin{array}{l} S_{Y\times Z} = - \quad \Rightarrow \quad S_Z = -S_Y = - \\ E_{Y\times Z} = E_Y + E_Z - 127 \Leftrightarrow 131 = 130 + E_Z - 127 \Leftrightarrow E_Z = 128 \\ M_{Y\times Z} \equiv M_Y \quad \Rightarrow \quad M_Z = 1{,}0 \end{array}$$

Logo,

$$Z = \underbrace{1}_{S_Z}\ \underbrace{10000000}_{E_Z}\ \underbrace{00...0}_{f_Z}$$

Exercício 6

Nesta questão, todos os números reais estão representados em vírgula flutuante (formato de precisão simples da norma IEEE 754).

a) Determine o número X cuja representação é dada por $01000010000111010000000000000000_2$.

b) Considerando um segundo número Y em que os 23 bits da sua mantissa são nulos, calcule:

   i) a mantissa de $X \times Y$;

   ii) o expoente de Y, sabendo que o expoente resultante de $X \times Y$, representado nos seus 8 bits, é 10000000_2.

a)

$$X = \underbrace{0}_{+} \quad \underbrace{10000100}_{E_X = 132} \quad \underbrace{00111010000000000000000}_{M_X = 1{,}0011101}$$

O expoente real de X é:

$$E_X^{\text{real}} = 132 - 127 = 5$$

Logo,

$$X_{10} = +1{,}0011101_2 \times 2^5 = 100111{,}01_2 = 39{,}25_{10}$$

b) i) Como os 23 bits da parte representável (parte fracionária) da mantissa são nulos, conclui-se que $M_Y = 1{,}f_Y = 1{,}0$. Assim,

$$M_{X\times Y} = M_X \times M_Y = M_X = 1{,}0011101_2$$

ii) O expoente do produto de dois números em vírgula flutuante é dado pela soma dos expoentes dos operandos menos o excesso 127. O expoente assim calculado nem sempre constitui o expoente definitivo do resultado, pois se o produto das mantissas não resultar normalizado então o expoente deve ser ajustado de acordo com a normalização. Porém, neste exercício não acontece tal situação porque é dito que $M_Y = 1{,}0$.
Assim,

$$\begin{aligned} E_{X\times Y} &= E_X + E_Y - 127 \\ 128 &= 132 + E_Y - 127 \\ E_Y &= 123 \quad (E_Y^{\text{real}} = -4) \end{aligned}$$

2.2 Exercícios propostos

Exercício 7

Considere que $11000001101100000000000000000000_2$ é a representação de um número em vírgula flutuante segundo a norma IEEE 754. Determine:

a) a mantissa do número;

b) o expoente do número;

c) o valor decimal representado.

Exercício 8

Represente em vírgula flutuante, no formato de precisão simples (32 bits) da norma IEEE 754, os seguintes números:

a) 31,25 b) $-0{,}625$ c) 0 d) 1026,5

Exercício 9

Considere a representação IEEE-754 de precisão simples. (Nota: use um conversor — por exemplo, `http://www.h-schmidt.net/FloatConverter/IEEE754.html` — para obter os números em decimal).

a) Qual é o mais pequeno número positivo (normalizado) representável?

b) Qual é o maior número normalizado representável?

c) Qual é o maior número negativo (normalizado) representável?

d) Qual é o menor número normalizado representável?

Exercício 10

Considere uma representação em vírgula flutuante normalizada do tipo IEEE-754, com 8 bits no total. O bit mais significativo representa o sinal e é seguido por um expoente de 3 bits (representado por notação *em excesso*) e um significando de 4 bits.

a) Qual é o valor representado por 1 001 1101?

b) Qual é o valor representado por 0 101 1001?

c) Qual é a representação de 3/8 neste formato?

Exercício 11

Considere os números decimais $A = 33$ e $B = -2{,}875$.

a) Represente A e B em vírgula flutuante, no formato de IEEE 754 de 32 bits.

b) Efetue as operações seguintes em vírgula flutuante no formato de 32 bits:

tsk[4] $A + B$ tsk[5] $B - A$ tsk[6] $A \times B$

c) Represente os resultados das operações em decimal e verifique se correspondem ao valores esperados.

Exercício 12

Dois números V_1 e V_2 estão representados em vírgula flutuante no formato de 32 bits. Os seus valores, expressos em hexadecimal, são:

V_1: $421D0000_{\text{H}}$ V_2: $C0000000_{\text{H}}$

Calcule:

a) $-V_2$ b) $V_1 + V_2$ c) $V_2 - V_1$ d) $V_1 \times V_2$

Exercício 13

Seja um número real X, cuja representação em vírgula flutuante (norma IEEE 754, com 32 bits) é $3F400000_{\text{H}}$. Considere também $Y = 11{,}625_{10}$.

a) Apresente em binário a representação de Y no mesmo formato.

b) Calcule $X + Y$ em vírgula flutuante, indicando claramente todos os passos efetuados.

Exercício 14

Considere a representação em vírgula flutuante, norma IEEE 754, com 32 bits. Sendo A um número real, cuja representação nesse formato é 40400000_{H}, e sendo $B = -10{,}25_{10}$:

a) apresente a representação binária de B no mesmo formato.

b) calcule $A - B$ em vírgula flutuante, indicando claramente todos os passos efetuados. No final, converta o resultado para decimal.

Exercício 15

Considere que S e T representam dois números em vírgula flutuante no formato de precisão simples definido pela norma IEEE 754:

S: $11000001010000000000000000000000_2$ T: $11000001101100000000000000000000_2$

a) Indique a representação de S em hexadecimal.

b) Mostre como se realiza a adição de S e T, indicando claramente todos os passos efetuados.

Exercício 16

No formato de precisão simples da norma IEEE 754, os números X e Y são representados por

X: $C3800000_{\text{H}}$ e Y: $00111111100000000000000000000000_2$.

Das afirmações seguintes indique a correta, fundamentando a sua escolha.

A. O expoente real de X é 8.
B. $X - Y$ é um número positivo.
C. O expoente real de X é 8 e Y é um número negativo.
D. $X > Y$.

Exercício 17

Em aplicações de aprendizagem por computador tem vindo a tornar-se frequente o uso do formato `Bfloat16`, que usa 8 bits para representar o expoente e 7 bits para a mantissa. São usadas as regras gerais da IEEE 754 com arredondamento (no caso de empate, arredondar para o número par representável mais próximo). Todos os números não-normalizados são tratados como zero.

a) Determinar o valor decimal do maior número representável.

b) Determinar o valor decimal do menor número positivo representável.

c) Determinar o resultado do cálculo $(2{,}6 \times 10^6) \times (3{,}1 \times 10^4)$ quando todos os números são representados em `Bfloat16`. Comente o resultado.

3 Circuitos combinatórios

3.1 Exercícios resolvidos

Exercício 1
Simplifique algebricamente as seguintes funções booleanas usando teoremas da álgebra de Boole.

a) $F(A,B,C) = A \cdot B + \overline{\overline{A} + B} + A \cdot C$.

b) $F(A,B,C) = (A+B+C) \cdot (A+B+\overline{C}) \cdot (\overline{A}+B+C) \cdot (\overline{A}+B+\overline{C})$.

c) $F(X,Y,Z) = X + \overline{X} \cdot Z + X \cdot \overline{Y}$.

d) $G(X,Y,Z) = X \cdot \overline{Y} \cdot Z + X \cdot \overline{Y} \cdot \overline{Z} + \overline{X} \cdot \overline{Y} \cdot \overline{Z}$.

e) $F(A,B,C) = A \cdot (\overline{B} + C) + B \cdot \overline{C}$.

a)
$$\begin{aligned} F(A,B,C) &= A \cdot B + \overline{\overline{A} + B} + A \cdot C \\ &= A \cdot B + A \cdot \overline{B} + A \cdot C \\ &= A \cdot (B + \overline{B}) + A \cdot C \\ &= A + A \cdot C \\ &= A \cdot (1 + C) \\ &= A \end{aligned}$$

b)
$$\begin{aligned} F(A,B,C) &= (A+B+C) \cdot (A+B+\overline{C}) \cdot (\overline{A}+B+C) \cdot (\overline{A}+B+\overline{C}) \\ &= \big((A+B) + (C \cdot \overline{C})\big) \cdot \big((\overline{A}+B) + (C \cdot \overline{C})\big) \\ &= (A+B) \cdot (\overline{A}+B) \\ &= B + A \cdot \overline{A} \\ &= B \end{aligned}$$

c)
$$\begin{aligned} F(X,Y,Z) &= X + \overline{X} \cdot Z + X \cdot \overline{Y} \\ &= X \cdot (1 + \overline{Y}) + \overline{X} \cdot Z \\ &= X + \overline{X} \cdot Z \\ &= X + Z \end{aligned}$$

d) $$\begin{aligned} G(X,Y,Z) &= X\cdot\overline{Y}\cdot Z + X\cdot\overline{Y}\cdot\overline{Z} + \overline{X}\cdot\overline{Y}\cdot\overline{Z} \\ &= X\cdot\overline{Y}(Z+\overline{Z}) + \overline{X}\cdot\overline{Y}\cdot\overline{Z} \\ &= \overline{Y}\cdot(X+\overline{X}\cdot\overline{Z}) \\ &= \overline{Y}\cdot(X+\overline{Z}) \\ &= X\cdot\overline{Y} + \overline{Y}\cdot\overline{Z} \end{aligned}$$

e) $$\begin{aligned} F(A,B,C) &= A\cdot(\overline{B}+C) + B\cdot\overline{C} \\ &= A\cdot\overline{\overline{(\overline{B}+C)}} + B\cdot\overline{C} \\ &= A\cdot(B\cdot\overline{C}) + B\cdot\overline{C}\cdot(A+\overline{A}) \\ &= A\cdot\overline{B}\cdot\overline{C} + A\cdot B\cdot\overline{C} + B\cdot\overline{C}\cdot\overline{A} + A\cdot B\cdot\overline{C} \\ &= A\cdot(\overline{B}\cdot\overline{C} + B\cdot\overline{C}) + B\cdot\overline{C}\cdot(\overline{A}+A) \\ &= A + B\cdot\overline{C} \end{aligned}$$

Exercício 2
Obtenha a tabela de verdade para cada uma das seguintes funções booleanas.

a) $F(A,B,C) = (A+\overline{B})\cdot(A+B+\overline{C})$

b) $F(X,Y,Z) = X\cdot\overline{Y} + \overline{X}\cdot Y\cdot\overline{Z}$

a) Se um termo soma é 0 então a função também é 0. Assim:

- se A=0 e B=1, então F=0;
- se A=0 e B=0 e C=1, então F=0.

Desta forma identificam-se as combinações das variáveis, isto é, as linhas da tabela de verdade onde F=0, tal como apresentado. Nas restantes situações F=1.

A	B	C	F		
0	0	0	1		
0	0	1	0	$\longleftarrow$	$A+B+\overline{C}$
0	1	0	0	$\longleftarrow$	$A+\overline{B}$
0	1	1	0	$\longleftarrow$	$A+\overline{B}$
1	0	0	1		
1	0	1	1		
1	1	0	1		
1	1	1	1		

b) Se um termo produto é 1 então a função também é 1. Assim:

- se X=1 e Y=0, então F=1;
- se X=0 e Y=1 e Z=0, então F=1.

Desta forma identificam-se as combinações das variáveis, isto é, as linhas da tabela de verdade onde F=1, tal como apresentado. Nas restantes situações F=0.

X	Y	Z	F		
0	0	0	0		
0	0	1	0		
0	1	0	1	$\longleftarrow$	$\overline{X}\cdot Y\cdot\overline{Z}$
0	1	1	0		
1	0	0	1	$\longleftarrow$	$X\cdot\overline{Y}$
1	0	1	1	$\longleftarrow$	$X\cdot\overline{Y}$
1	1	0	0		
1	1	1	0		

Exercício 3

Escreva uma expressão booleana para as funções lógicas representadas pelas tabelas de verdade indicadas.

a)

x	y	f
0	0	1
0	1	0
1	0	0
1	1	1

b)

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

A partir da tabela de verdade de uma função booleana podem retirar-se expressões algébricas na forma de uma soma de produtos ou na forma de um produto de somas.

Para a expressão na forma de uma *soma de produtos* identificam-se as linhas da tabela de verdade onde a função é 1 e forma-se um termo produto tal que nessa combinação das variáveis da função o valor do produto seja 1 (igual ao valor da função). A expressão da função obtém-se pela soma de todos os termos produto nestas condições.

Para a expressão na forma de um *produto de somas* identificam-se as linhas da tabela de verdade onde a função é 0 e forma-se um termo soma tal que nessa combinação das variáveis da função o valor da soma seja 0 (igual ao valor da função). A expressão da função obtém-se pelo produto de todos os termos soma nestas condições.

A partir destas expressões podem obter-se outras equivalentes por simplificação baseada em teoremas da álgebra de Boole.

a) Optando pela forma soma de produtos e procedendo da forma descrita, os termos produto que definem a função são os indicados.

x	y	f		
0	0	1	$\longrightarrow$	$\overline{x}\cdot\overline{y}$
0	1	0		
1	0	0		
1	1	1	$\longrightarrow$	$x\cdot y$

Logo, $f(x,y) = \overline{x}\cdot\overline{y} + x\cdot y$.

b) Optando pela forma produto de somas e procedendo da forma descrita, os termos soma que definem a função são os indicados.

x	y	z	g		
0	0	0	1		
0	0	1	1		
0	1	0	1		
0	1	1	0	$\longrightarrow$	$x + \overline{y} + \overline{z}$
1	0	0	1		
1	0	1	1		
1	1	0	0	$\longrightarrow$	$\overline{x} + \overline{y} + z$
1	1	1	0	$\longrightarrow$	$\overline{x} + \overline{y} + \overline{z}$

Logo, $g(x,y,z) = (x + \overline{y} + \overline{z})\cdot(\overline{x} + \overline{y} + z)\cdot(\overline{x} + \overline{y} + \overline{z})$.

Exercício 4

Considere a função booleana $F(X,Y,Z)$ com

$$F(X,Y,Z) = \overline{X}\cdot\overline{Y}\cdot\overline{Z} + \overline{X}\cdot\overline{Y}\cdot Z + \overline{X}\cdot\ Y\cdot\overline{Z} + X\cdot\overline{Y}\cdot\overline{Z} + X\cdot\overline{Y}\cdot Z + X\cdot Y\cdot Z$$

a) Simplifique a expressão de $F(X,Y,Z)$.

b) Construa a tabela de verdade da função.

c) Indique a expressão de $F(X,Y,Z)$ na forma de um produto de somas.

d) Obtenha um circuito lógico que realiza a função $F(X,Y,Z)$, usando apenas portas lógicas do tipo NOR.

a)
$$\begin{aligned} F(X,Y,Z) &= \overline{X}\cdot\overline{Y}\cdot\overline{Z} + \overline{X}\cdot\overline{Y}\cdot Z + \overline{X}\cdot Y\cdot\overline{Z} + X\cdot\overline{Y}\cdot\overline{Z} + X\cdot\overline{Y}\cdot Z + X\cdot Y\cdot Z \\ &= \overline{X}\cdot\overline{Y}\cdot(\overline{Z} + Z) + \overline{X}\cdot Y\cdot\overline{Z} + X\cdot\overline{Y}\cdot(\overline{Z} + Z) + X\cdot Y\cdot Z \\ &= \overline{X}\cdot\overline{Y} + X\cdot\overline{Y} + \overline{X}\cdot Y\cdot\overline{Z} + X\cdot Y\cdot Z \\ &= \overline{Y} + \overline{X}\cdot Y\cdot\overline{Z} + X\cdot Y\cdot Z \\ &= \overline{Y} + \overline{X}\cdot\overline{Z} + X\cdot Z \end{aligned}$$

b) Como a expressão de F está na forma de uma soma de produtos, cada termo produto identifica combinações das entradas X, Y, Z onde $F = 1$. Ao lado da tabela de verdade indicam-se esses termos.

X	Y	Z	F		
0	0	0	1	$\longleftarrow$	$\overline{Y}$ e $\overline{X}\cdot\overline{Z}$
0	0	1	1	$\longleftarrow$	$\overline{Y}$
0	1	0	1	$\longleftarrow$	$\overline{X}\cdot\overline{Z}$
0	1	1	0		
1	0	0	1	$\longleftarrow$	$\overline{Y}$
1	0	1	1	$\longleftarrow$	$\overline{Y}$ e $X\cdot Z$
1	1	0	0		
1	1	1	1	$\longleftarrow$	$X\cdot Z$

c) Na tabela de verdade identificam-se os termos soma para os quais F=0. São eles

$$X+\overline{Y}+\overline{Z} \qquad e \qquad \overline{X}+\overline{Y}+Z$$

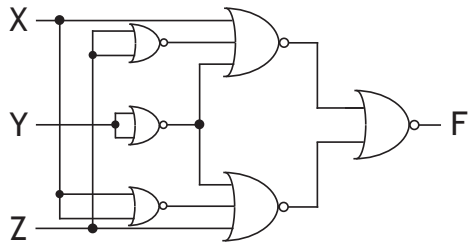
resultando

$$F(X,Y,Z)=(X+\overline{Y}+\overline{Z})\cdot(\overline{X}+\overline{Y}+Z)$$

d) Considerando a expressão da função na forma de um produto de somas, pode obter-se uma expressão equivalente apenas com somas lógicas negadas, cada uma das quais será realizada por uma porta NOR no circuito lógico pretendido. A negação das variáveis pode também realizar-se através de um NOR aplicando a variável a negar a ambas as entradas do NOR.

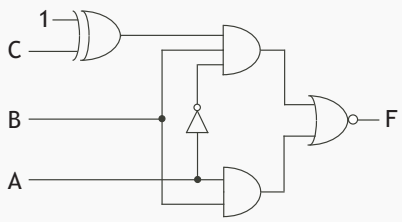
$$F(X,Y,Z)=\overline{\overline{(X+\overline{Y}+\overline{Z})\cdot(\overline{X}+\overline{Y}+Z)}}=\overline{\overline{X+\overline{Y}+\overline{Z}}+\overline{\overline{X}+\overline{Y}+Z}}$$

O circuito resultante é:



Exercício 5

O circuito da figura implementa uma função $F(A,B,C)$.



a) Deduza uma expressão da função lógica realizada pelo circuito, indicando-a na forma de um produto de somas.

b) Escreva $F(A,B,C)$ como uma soma de produtos simplificada.

a) A expressão da função realizada pelo circuito lógico pode obter-se através das expressões que resultam da saída de cada porta lógica como funções das entradas do circuito.

$$\begin{aligned} F(A,B,C) &= \overline{\overline{A}\cdot B\cdot(1\oplus C)+A\cdot B} \\ &= \overline{\overline{A}\cdot B\cdot\overline{C}+A\cdot B} \\ &= \overline{\overline{A}\cdot B\cdot\overline{C}}\cdot\overline{A\cdot B} \\ &= (A+\overline{B}+C)\cdot(\overline{A}+\overline{B}) \end{aligned}$$

b)
$$\begin{aligned} F(A,B,C) &= (A+\overline{B}+C)\cdot(\overline{A}+\overline{B}) \\ &= A\cdot\overline{A}+A\cdot\overline{B}+\overline{A}\cdot\overline{B}+\overline{B}+\overline{A}\cdot C+\overline{B}\cdot C \\ &= (A+\overline{A}+1+C)\cdot\overline{B}+\overline{A}\cdot C \\ &= \overline{B}+\overline{A}\cdot C \end{aligned}$$

Exercício 6

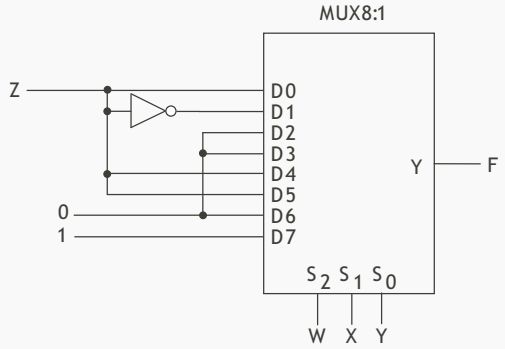
Pretende-se projetar um circuito capaz de detetar se um número com 3 bits $n_2n_1n_0$ aplicado à sua entrada está compreendido entre 2 e 5 (inclusive). A saída do circuito é uma função de 3 variáveis, $G(n_2, n_1, n_0)$, sendo $G = 1$ para os números nas condições indicadas e $G = 0$ no caso contrário. Defina o comportamento do circuito que realiza G na forma de uma tabela de verdade.

Para os números compreendidos entre 2 e 5, formados em binário por $n_2n_1n_0$, $G = 1$. Para os restantes G é 0. Assim resulta a tabela de verdade seguinte:

n_2	n_1	n_0	G
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Exercício 7

A figura mostra um circuito, baseado num multiplexador de 8 para 1, que realiza uma função $F(W, X, Y, Z)$.



a) Defina $F(W, X, Y, Z)$ através de uma tabela de verdade.

b) Represente $F(W, X, Y, Z)$ através de uma expressão algébrica.

a) As variáveis W, X e Y determinam a entrada do multiplexador que é selecionada. O valor nela aplicado surge na saída do circuito. A tabela de verdade pretendida obtém-se considerando todas as combinações das variáveis e consequentes valores da função.

W	X	Y	Z	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

b) Considere-se a forma soma de produtos nesta resolução. Embora não seja requerido no enunciado vai proceder-se à simplificação da expressão.

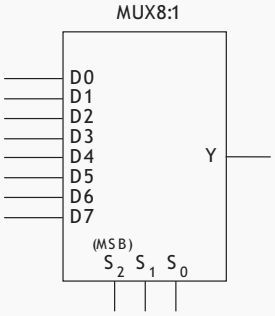
$$
\begin{aligned}
F &= \overline{W}\cdot\overline{X}\cdot\overline{Y}\cdot Z + \overline{W}\cdot\overline{X}\cdot Y\cdot\overline{Z} + W\cdot\overline{X}\cdot\overline{Y}\cdot Z + W\cdot\overline{X}\cdot Y\cdot Z + W\cdot X\cdot Y\cdot\overline{Z} + W\cdot X\cdot Y\cdot Z \\
&= \overline{W}\cdot\overline{X}\cdot\overline{Y}\cdot Z + \overline{W}\cdot\overline{X}\cdot Y\cdot\overline{Z} + W\cdot\overline{X}\cdot Z\cdot(\overline{Y}+Y) + W\cdot X\cdot Y\cdot(\overline{Z}+Z) \\
&= \overline{W}\cdot\overline{X}\cdot\overline{Y}\cdot Z + \overline{W}\cdot\overline{X}\cdot Y\cdot\overline{Z} + W\cdot\overline{X}\cdot Z + W\cdot X\cdot Y \\
&= \overline{X}\cdot(Z\cdot(\overline{W}\cdot\overline{Y}+W) + \overline{W}\cdot Y\cdot\overline{Z}) + W\cdot X\cdot Y \\
&= \overline{X}\cdot(\overline{Y}\cdot Z + W\cdot Z + \overline{W}\cdot Y\cdot\overline{Z}) + W\cdot X\cdot Y \\
&= \overline{X}\cdot\overline{Y}\cdot Z + W\cdot\overline{X}\cdot Z + \overline{W}\cdot\overline{X}\cdot Y\cdot\overline{Z} + W\cdot X\cdot Y
\end{aligned}
$$

Exercício 8

Seja a função booleana $S = (A + B + C) \cdot (A + \overline{B} + \overline{C})$.

a) Represente S através de uma tabela de verdade.

b) Realize a função S recorrendo ao multiplexador de 8 para 1 da figura.



a)

A	B	C	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

b) Efetuar as seguintes ligações:

D0 = 0, D1 = D2 = 1, D3 = 0,
D4 = D5 = D6 = D7 = 1,
S2 = A, S1 = B, S0 = C e $Y = F$.

Exercício 9

Pretende-se realizar um circuito capaz de comparar duas quantidades positivas A e B, representadas em binário com 2 bits cada uma (a_1a_0 e b_1b_0), e produzir duas saídas, X e Y. A saída X deve ser 1 se e só se $A = B$, e a saída Y deve ser 1 se e só se $A > B$.

a) Construa uma tabela de verdade de X e Y como funções de a_1, a_0, b_1 e b_0.

b) Obtenha um circuito que realize a função Y.

c) Mostre como, com um mínimo de esforço, poderia acrescentar a este circuito uma saída Z que fosse 1 quando $A < B$.

d) Admitindo que tinha disponíveis vários circuitos como o descrito atrás, mostre como os poderia utilizar para realizar a comparação de quantidades de 6 bits cada, isto é, de modo a detetar as situações de $A = B$ e $A > B$ quando $A = a_5a_4a_3a_2a_1a_0$ e $B = b_5b_4b_3b_2b_1b_0$.

a)

a_1	a_0	b_1	b_0	X	Y
0	0	0	0	1	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	1	0

b)
$$\begin{aligned}
Y &= \overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot\overline{b_0} + a_1\cdot\overline{a_0}\cdot\overline{b_1} + a_1\cdot a_0\cdot\overline{b_1} + a_1\cdot a_0\cdot b_1\cdot\overline{b_0} \\
&= \overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot\overline{b_0} + a_1\cdot\overline{b_1} + a_1\cdot a_0\cdot b_1\cdot\overline{b_0} \\
&= \overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot\overline{b_0} + a_1\cdot(\overline{b_1} + a_0\cdot b_1\cdot\overline{b_0}) \\
&= \overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot\overline{b_0} + a_1\cdot(\overline{b_1} + a_0\cdot\overline{b_0}) \\
&= a_1\cdot\overline{b_1} + a_1\cdot a_0\cdot\overline{b_0} + \overline{a_1}\cdot a_0\cdot\overline{b_1}\cdot\overline{b_0} \\
&= a_1\cdot\overline{b_1} + a_0\cdot\overline{b_0}(a_1 + \overline{a_1}\cdot\overline{b_1}) \\
&= a_1\cdot\overline{b_1} + a_0\cdot\overline{b_0}(a_1 + \overline{b_1}) \\
&= a_1\cdot\overline{b_1} + a_0\cdot\overline{b_1}\cdot\overline{b_0} + a_1\cdot a_0\cdot\overline{b_0}
\end{aligned}$$

Desenhar o circuito a partir da expressão encontrada.

c) A função pretendida define-se como
$$Z = \overline{X}\cdot\overline{Y}$$

d) Usar 3 circuitos comparadores de 2 bits idênticos aos da alínea a), combinando as saídas da seguinte forma:
$$X = X_1\cdot X_2\cdot X_3 \qquad \text{e} \qquad Y = Y_3 + X_3\cdot Y_2 + X_3\cdot X_2\cdot Y_1$$
em que:

- X_1 é a saída X do comparador de a_1a_0 com b_1b_0;
- X_2 é a saída X do comparador de a_3a_2 com b_3b_2;
- X_3 é a saída X do comparador de a_5a_4 com b_5b_4;
- Y_1, Y_2 e Y_3 são as saídas Y dos comparadores correspondentes.

3.2 Exercícios propostos

Exercício 10

Simplifique algebricamente as seguintes funções booleanas utilizando teoremas da álgebra de Boole.

a) $F(A,B,C,D,E) = A \cdot B \cdot \overline{C} + \overline{C} \cdot \overline{D} \cdot E + A \cdot B + A \cdot B \cdot \overline{C} \cdot \overline{D} \cdot E + A \cdot B \cdot D \cdot \overline{E} + \overline{C} \cdot D \cdot E$.

b) $F(A,B,C) = \overline{A + A \cdot \overline{B} + \overline{A} \cdot C}$.

c) $G(A,B,C) = A \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot \overline{C}$.

d) $F(A,B,C,D) = \overline{B} \cdot C \cdot \overline{D} + \overline{A} \cdot (\overline{\overline{C} + B}) + A \cdot C \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot D$.

e) $F(W,X,Y,Z) = \overline{\overline{W} \cdot (\overline{X} + Y \cdot (Z + W))}$.

f) $F(A,B,C,D) = \overline{A} \cdot B \cdot C + \overline{B} \cdot C \cdot D + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + A \cdot B \cdot C \cdot D$.

Exercício 11

Obtenha a tabela de verdade para cada uma das seguintes funções booleanas.

a) $F(A,B,C) = A \cdot \overline{B} + \overline{A} \cdot \overline{C}$.

b) $G(X,Y,Z) = (X + \overline{Z}) \cdot (\overline{X} + \overline{Y} + Z)$.

c) $F(W,X,Y,Z) = \overline{W \cdot X} \cdot (\overline{\overline{Y} + \overline{Z}})$.

Exercício 12

Considere a função $F(X,Y,Z) = X \cdot \overline{Y} + X \cdot Y \cdot \overline{Z} + \overline{X} \cdot Z$.

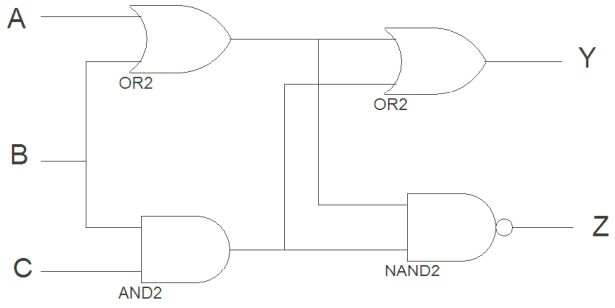
a) Indique a respetiva tabela de verdade.

b) Escreva $F(X,Y,Z)$ como um produto de somas.

c) Desenhe o circuito lógico correspondente.

Exercício 13

Considere o circuito lógico apresentado na figura.



a) Deduza a expressão booleana simplificada correspondente à saída Y do circuito.

b) Mostre que a saída Z pode ser definida por $Z(A,B,C) = \overline{B} + \overline{C}$.

Exercício 14

Considere a função booleana $F(A,B,C) = \overline{A + \overline{B + \overline{C}}} + \overline{\overline{A}\cdot B}\cdot C$.

a) Indique a expressão de $F(A,B,C)$ como uma soma de produtos simplificada.

b) Construa a tabela de verdade da função $F(A,B,C)$.

c) Obtenha um circuito lógico que realize a função $F(A,B,C)$.

Exercício 15

Considere as seguintes funções booleanas:

$$F = \overline{X}\cdot Y + \overline{X}\cdot\overline{Y}\cdot Z \quad \text{e} \quad G = (\overline{A} + B + C)\cdot(A + \overline{B} + \overline{D})\cdot(B + \overline{C} + \overline{D})\cdot(A + B + C + D).$$

a) Represente $F(X,Y,Z)$ através de uma tabela de verdade e obtenha uma expressão na forma de um produto de somas.

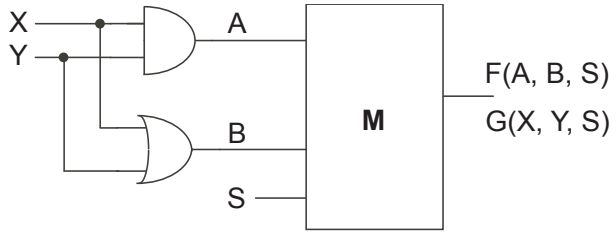
b) Represente $G(A,B,C,D)$ através de uma tabela de verdade e obtenha uma expressão na forma de uma soma de produtos (não simplificada).

Exercício 16

Seja a função booleana $G(A,B,C) = A\cdot\overline{C} + \overline{B}\cdot A\cdot C + B\cdot C\cdot\overline{A}$. Exprima $G(A,B,C)$ na forma de um produto de somas simplificado.

Exercício 17

A figura seguinte mostra um circuito que realiza uma função $G(X,Y,S)$. Além de portas lógicas, o circuito inclui um bloco, M, que realiza a função $F(A,B,S)$ definida por: $F = A$ se $S = 0$ e $F = B$ se $S = 1$.



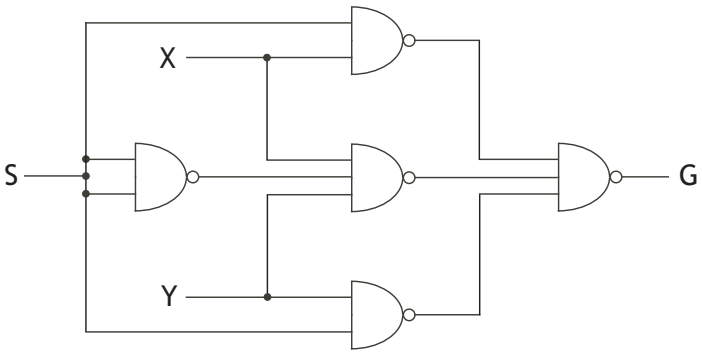
a) Exprima a função F numa tabela de verdade.

b) Indique uma expressão simplificada para $F(A,B,S)$.

c) O circuito que realiza a função F é um multiplexador (*multiplexer*) de 2 para 1. Mostre como é constituído.

d) Encontre uma expressão simplificada do tipo soma de produtos para a função $G(X,Y,S)$ realizada pelo circuito.

e) Mostre que o circuito seguinte, usando apenas portas NAND, realiza a função G.



Exercício 18

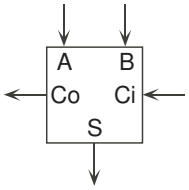
Considere um circuito que apresenta na saída S o valor lógico 1 sempre que na sua entrada o número positivo de 4 bits, $A_3A_2A_1A_0$, é maior que 5 e múltiplo de 4.

a) Construa a tabela de verdade da função $S(A_3, A_2, A_1, A_0)$.

b) Obtenha uma expressão para $S(A_3, A_2, A_1, A_0)$ e simplifique-a.

Exercício 19

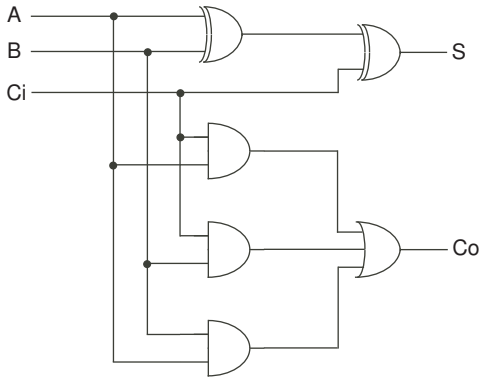
Um circuito elementar utilizado na construção de circuitos digitais para aritmética binária é o somador completo (*full-adder*) representado na figura. O somador tem 2 saídas Co e S que representam em binário a soma dos valores (0 ou 1) presentes nas entradas A, B e Ci.



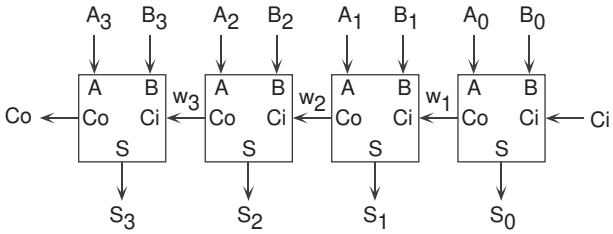
a) Construa a tabela de verdade correspondente às funções $S(A, B, Ci)$ e $Co(A, B, Ci)$.

b) Escreva a expressão das funções $S(A, B, Ci)$ e $Co(A, B, Ci)$ na forma de uma soma de produtos.

c) Verifique que o circuito lógico da figura seguinte realiza as funções $S(A, B, Ci)$ e $Co(A, B, Ci)$.



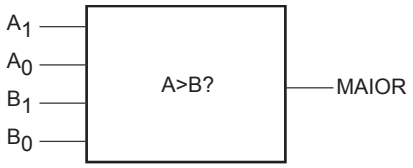
d) Considere agora um somador de 4 bits, constituído por 4 somadores completos, como mostra a figura.



Identifique na figura o valor de cada um dos sinais admitindo que os valores a somar, representando números positivos, são $A = A_3A_2A_1A_0 = 1010$ e $B = B_3B_2B_1B_0 = 1110$.

Exercício 20

Pretende-se construir um circuito combinatório para comparar dois números de 2 bits, sem sinal, $A = A_1A_0$ e $B = B_1B_0$. A saída $MAIOR$ é 1 quando A for maior do que B e 0 no caso contrário.

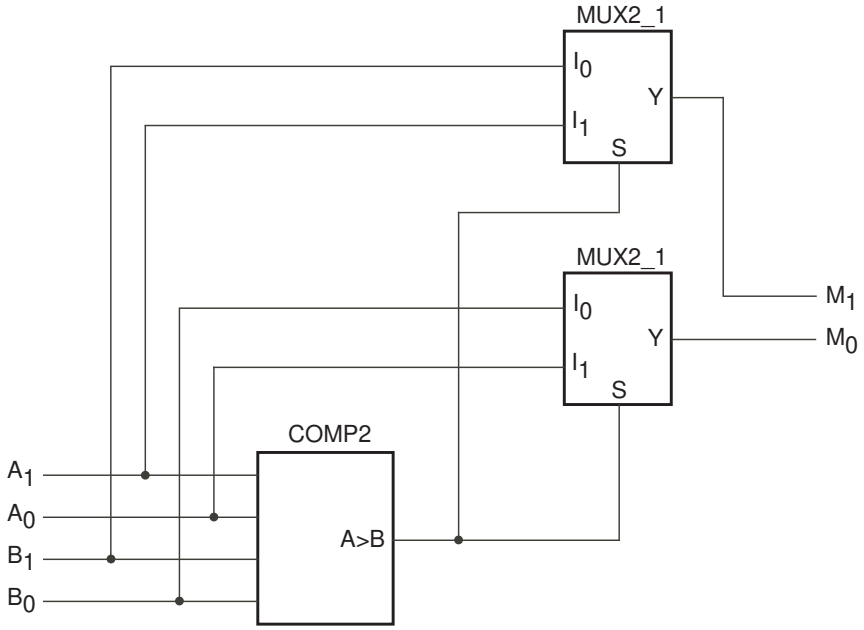


a) Expresse a função do circuito numa tabela de verdade.

b) Escreva uma expressão da função $MAIOR(A_1, A_0, B_1, B_0)$.

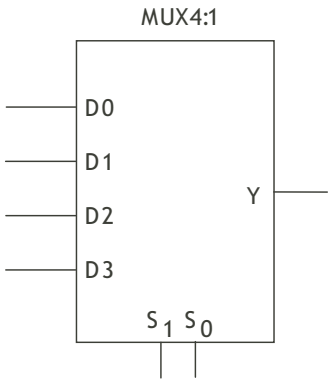
Exercício 21

O circuito da figura contém um comparador de magnitude de 2 bits e 2 multiplexadores de 2 para 1. As entradas do circuito formam dois números positivos, de 2 bits, $A = A_1A_0$ e $B = B_1B_0$. As saídas definem um número, também com 2 bits, $M = M_1M_0$. Analise o circuito e identifique a sua funcionalidade.



Exercício 22

Um multiplexador com n entradas de seleção pode ser usado para implementar qualquer função lógica de n variáveis. Na figura mostra-se o símbolo de um multiplexador de 4 para 1, o qual possui 2 entradas de seleção.



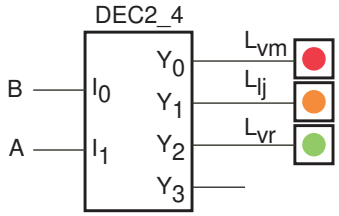
a) Mostre como implementar o produto lógico de duas variáveis recorrendo ao multiplexador.

b) Realize a função $F(X,Y) = X \cdot \overline{Y} + \overline{X} \cdot Y$ com o multiplexador apresentado.

c) Verifique, exemplificando, que é igualmente possível implementar funções de três variáveis. Sugestão: represente uma função de 3 variáveis através de uma tabela de verdade e, para cada combinação das variáveis, relacione o valor da função com uma dessas variáveis.

Exercício 23

A figura mostra um circuito constituído por um descodificador binário de 2 para 4 e um conjunto de 3 lâmpadas. O estado das lâmpadas é controlado pelas entradas A e B do circuito, ou seja, L_{vm}, L_{lj} e L_{vr} são funções de A e B. Admita que para ligar uma lâmpada é necessário que a saída do descodificador que a controla tenha o valor lógico 1.



a) Indique o estado de cada lâmpada se $AB = 01$ e $AB = 11$.

b) Determine o valor das entradas de modo a ligar, simultaneamente, as lâmpadas verde ($L_{vr} = 1$) e laranja ($L_{lj} = 1$).

Exercício 24

O descodificador binário n-para-2^n é um circuito muito comum.

a) Mostrar como construir um descodificador binário 2-4 usando portas lógicas simples.

b) Mostrar como acrescentar uma entrada de habilitação *(enable)* ao circuito anterior.

c) Mostrar como construir um descodificador 3-para-8 (sem *enable)* a partir de dois descodificadores 2-4 com entrada *enable*.

d) Mostrar como acrescentar uma entrada de habilitação (*enable*) ao circuito anterior.

Exercício 25

Usar um descodificador binário 3-para-8 (entradas $I_0 \ldots I_2$, saídas $Q_0 \ldots Q_7$) nas alíneas que se seguem.

a) Escrever as funções lógicas das saídas Q_0 e Q_3 do descodificador.

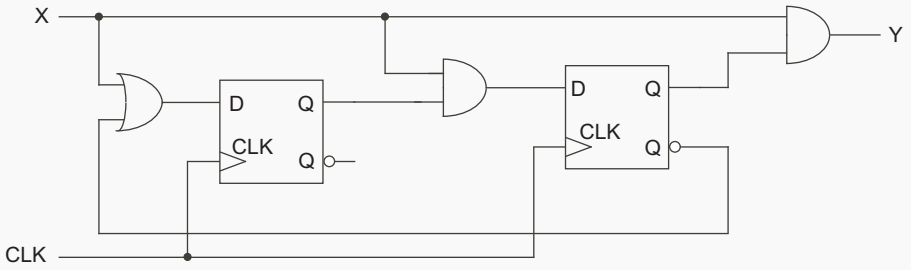
b) Mostrar como implementar a função booleana $F(X, Y, Z) = X \cdot Y \cdot \overline{Z} + \overline{X} \cdot Z$ usando apenas um descodificador e portas lógicas do tipo OR.

4 Circuitos sequenciais

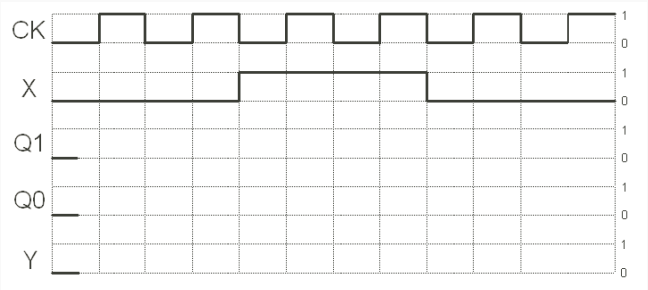
4.1 Exercícios resolvidos

Exercício 1

Considere o seguinte circuito com uma entrada X e uma saída Y.

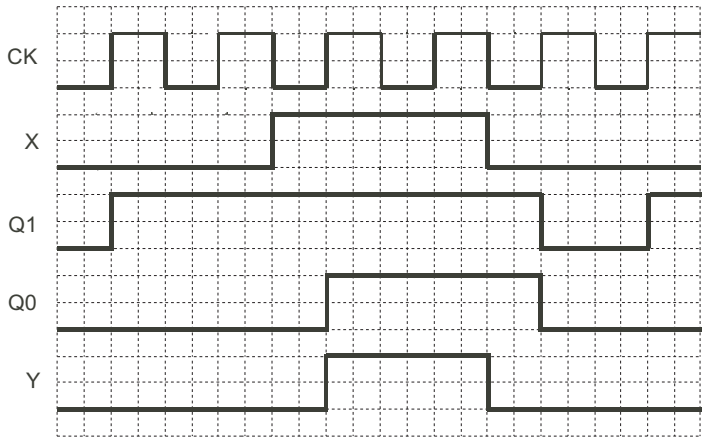


a) Desenhe no diagrama seguinte a evolução temporal das saídas dos *flip-flops* (Q_1 e Q_0), assim como da saída Y, sabendo que a entrada X evolui da forma representada. Nota: considere que no instante inicial Q_1 e Q_0 têm o valor lógico 0.



b) Admitindo que o período do sinal de relógio é 20 ns, indique ao fim de quanto tempo a saída Y passa de 0 a 1 pela primeira vez.

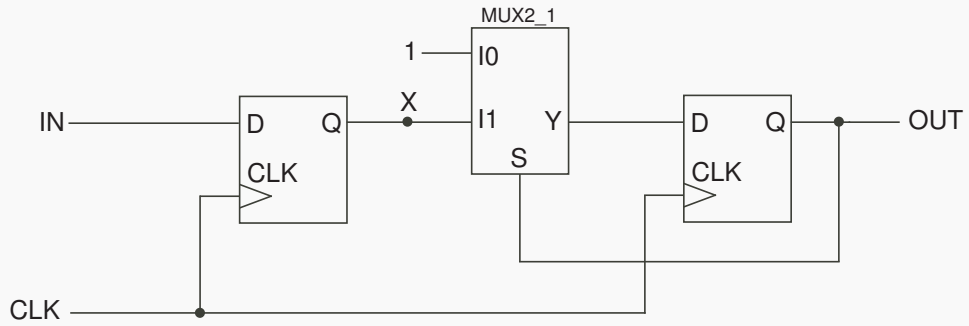
a) Como inicialmente $Q_0 = 0$ e $Q_1 = 0$, nas entradas dos *flip-flops* vão estar os valores $D_0 = 0$ e $D_1 = 1$, uma vez que $X = 0$. Ao ocorrer a primeira transição do sinal de relógio são estes os valores capturados pelos *flip-flops*, aparecendo nas saídas respetivas. Na próxima transição, o estado das saídas e X determinam os novos valores que os *flip-flops* vão apresentar. Assim se completam as formas de onda apresentadas.



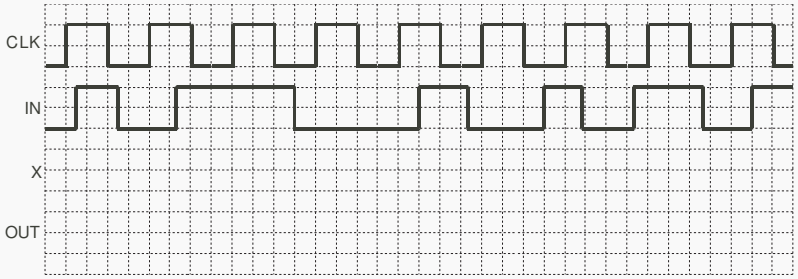
b) Considerando $T = 20\,\text{ns}$, verifica-se pelo resultado da alínea anterior que Y transita de 0 para 1 ao fim de $2{,}5 \times T$ =50 ns.

Exercício 2

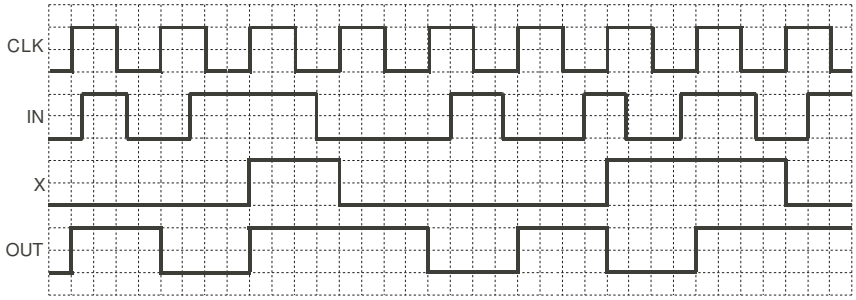
Considere o seguinte circuito sequencial, constituído por dois *flip-flops* e um multiplexador. Os *flip-flops* são sensíveis à transição ascendente do sinal de relógio (CLK) e o seu estado inicial é 0.



Assumindo a sequência de valores da entrada do circuito (IN) indicada na figura seguinte, apresente a forma de onda dos sinais X e OUT.

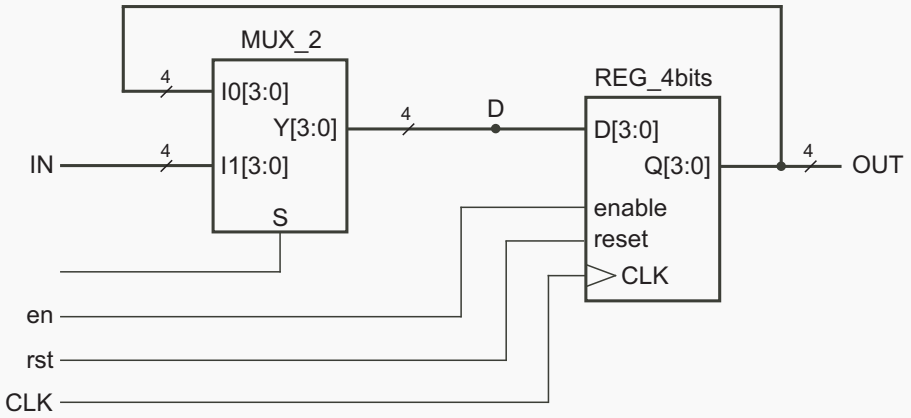


Inicialmente, X e OUT apresentam o valor 0. Sendo a entrada de seleção do multiplexador 0, o valor que surge na sua saída é 1, referente à entrada I_0. Como $IN = 0$ e $Y = 1$ quando ocorre a primeira transição do sinal de relógio, então X e OUT passam a assumir os valores 0 e 1, respetivamente. A análise do sucedido para as restantes transições segue o mesmo raciocínio, levando ao resultado apresentado na figura seguinte.



Exercício 3

Considere o circuito, composto por um registo de 4 bits e um multiplexador com duas entradas de 4 bits. Admita que as entradas *enable* e *reset* do registo são ativadas pelo valor lógico 1.



Nas alíneas seguintes, considere em cada transição ativa do sinal de relógio o valor das entradas apresentadas em cada tabela. Determine o valor da saída OUT após cada uma das transições assinaladas.

a)

Transição	D	en	rst
1	0110	1	0
2	0100	1	1
3	0100	0	0
4	1101	1	0

b)

Transição	IN	sel	en	rst
1	1001	1	1	0
2	1111	1	1	0
3	0101	0	1	0
4	1000	1	0	0
5	1000	1	1	0
6	1111	1	1	1
7	0101	0	1	0
8	0000	1	1	0
9	0001	1	1	0

a) Os sinais considerados só envolvem o registo de 4 bits. Para cada transição ativa do sinal de relógio é necessário ter em consideração que a saída do registo assume o valor presente na entrada D se a entrada de habilitação (*enable*) estiver ativa ($en = 1$), permanecendo inalterado até à próxima transição. Caso en seja 0, a saída mantém o valor anterior. Relativamente à

entrada *reset* do registo, quando ativo ($rst = 1$) coloca a saída em 0000, independentemente do valor das restantes entradas.

Aplicando estas considerações à sequência de entradas dada, obtém-se a saída OUT como se mostra na tabela seguinte.

Transição (CLK)	D	en	rst	OUT
1	0110	1	0	0110
2	0100	1	1	0000
3	0010	0	0	0000
4	1101	1	0	1101

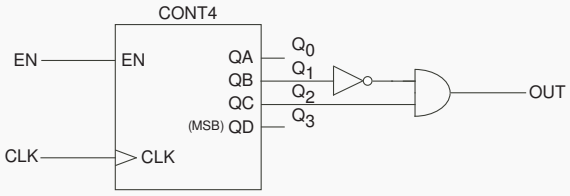
b) Além do que foi descrito na análise anterior, há agora que ter em consideração o multiplexador. Observando a forma como está a ser usado, conclui-se que o valor aplicado na entrada do registo provém da entrada IN, quando $sel = 1$, ou da saída OUT do registo, quando $sel = 0$.

Aplicando estas considerações à sequência de entradas dada, obtém-se a saída OUT como se mostra na tabela seguinte.

Transição (CLK)	IN	sel	en	rst	OUT
1	1001	1	1	0	1001
2	1111	1	1	0	1111
3	0101	0	1	0	1111
4	1000	1	0	0	1111
5	1000	1	1	0	1000
6	1111	1	1	1	0000
7	0101	0	1	0	0000
8	0000	1	1	0	0000
9	0001	1	1	0	0001

Exercício 4

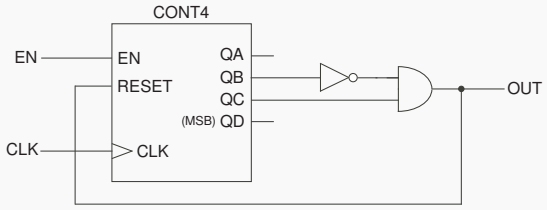
A figura representa um circuito sequencial baseado num contador de 4 bits.



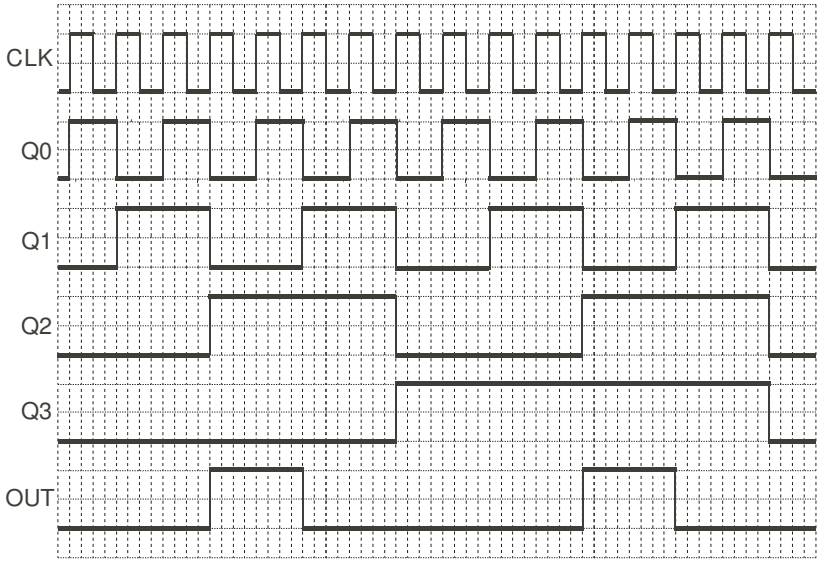
a) Desenhe as formas de onda das saídas do contador (Q_3, Q_2, Q_1 e Q_0) e do circuito completo (OUT).

b) Indique a sequência de estados ocorridos, numerando-os em decimal.

c) Considere agora que o contador possui uma entrada de *reset* (ativa a 1), acionada como mostra a figura seguinte. Determine a sequência de estados da saída do contador.



a) O circuito apresentado é constituído por um contador de 4 bits. Considerando que inicialmente as suas saídas são nulas, a cada transição do sinal de relógio o valor das saídas é incrementado, resultando a representação das formas de onda mostrada na figura seguinte.



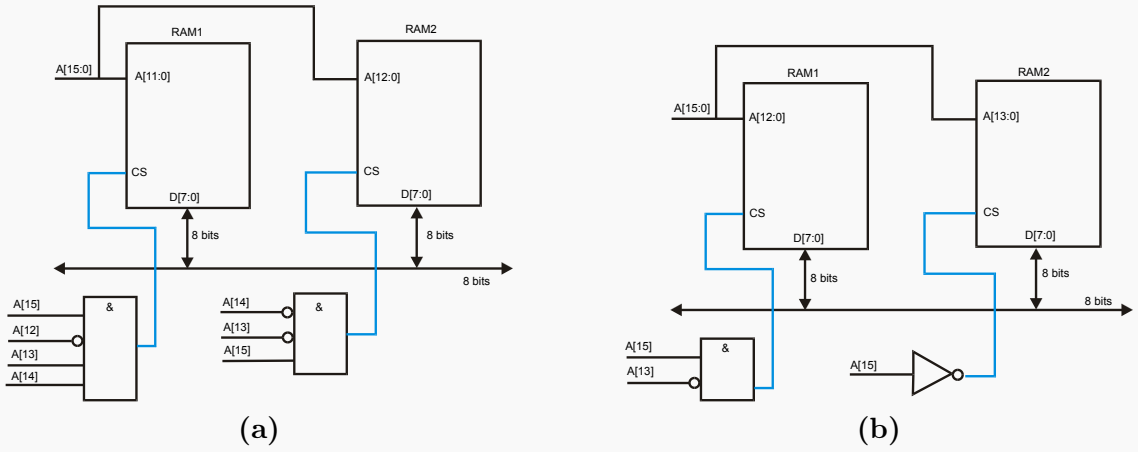
b) Sequência de estados (valores resultantes de $Q_3Q_2Q_1Q_0$):

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, \cdots$$

c) No estado $Q_DQ_CQ_BQ_A = 0100$ a entrada de *reset* fica ativa. Admitindo-o síncrono, na próxima transição do sinal de relógio o estado do contador passa para 0000. A sequência de estados resultante é: 0, 1, 2, 3, 4, 0, $\cdots$.

Exercício 5

Considere os sistemas de memória externa apresentados na figura seguinte.



a) Determine a capacidade de cada circuito de memória.

b) Elabore o mapa de memória de cada sistema. Indique, justificando, se a descodificação de memória é total ou parcial.

c) Para cada um dos seguintes endereços, indique o componente, ou seja, o circuito de memória, afetado: B7FF_H, 1000_H, E7AA_He 8000_H.

a) Obtém-se a capacidade de um circuito de memória multiplicando o número de posições pelo número de bits por posição. O número de bits por posição é igual ao número de bits do porto de dados `D[]`. Todos os circuitos deste problema têm um porto de dados de 8 bits (1 byte).

O número de posições é determinado pela dimensão do porto de endereços `A[]`. Para portos de N bits, existem 2^N posições.

Assim, para a figura (a) temos:

- RAM1: $2^{12} \times 8\,\text{bits} = 2^2 \times 2^{10} \times 8\,\text{bits} = 4 \times 2^{10}\,\text{byte} = 4\,\text{KiB}$
- RAM2: $2^{13} \times 8\,\text{bits} = 8\,\text{KiB}$

Para a figura (b) temos:

- RAM1: $2^{13} \times 8\,\text{bits} = 8\,\text{KiB}$
- RAM2: $2^{14} \times 8\,\text{bits} = 16\,\text{KiB}$

b) O mapa de memória indica, para cada endereço possível, qual o circuito que armazena os dados correspondentes. Em ambas as figuras, o barramento de endereços do CPU tem 16

bits (A[15:0]). Portanto, o espaço de endereçamento tem 2^{16} posições (de 1 byte, neste caso), i.e., 64 KB. A gama de endereços vai de $\texttt{0000}_\texttt{H}$ a $\texttt{FFFF}_\texttt{H}$.

Os endereços mapeados em cada circuito podem ser determinados por análise das condições em que o circuito está habilitado, i.e., para que endereços é que se tem CS=1.

Para a figura (a) temos:

- RAM1: $\text{CS} = A_{15} \cdot A_{14} \cdot A_{13} \cdot \overline{A_{12}} = 1$.
  Esta condição só é satisfeita se $A_{15} = 1, A_{14} = 1, A_{13} = 1, A_{12} = 0$.
  Logo, os endereços mapeadas na RAM1 têm o formato
  1110 XXXX XXXX XXXX
  em que X indica que o bit correspondente tanto pode ser 0 como 1. Os endereços com este formato estão na gama:
  1110 0000 0000 0000 a 1110 1111 1111 1111.
  Em hexadecimal, a gama é $\texttt{E000}_\texttt{H}$-$\texttt{EFFF}_\texttt{H}$.
  Como todos os bits do endereço são usados ($A_{15} - A_{12}$ na definição de CS; os restantes na ligação ao porto de endereços de RAM1), trata-se de *descodificação total*.
- RAM2: $\text{CS} = A_{15} \cdot \overline{A_{14}} \cdot \overline{A_{13}} = 1$.
  Esta condição só é satisfeita se $A_{15} = 1, A_{14} = 0, A_{13} = 0$.
  Logo, os endereços mapeadas na RAM2 têm o formato
  100X XXXX XXXX XXXX.
  Os endereços com este formato estão na gama:
  1000 0000 0000 0000 a 1001 1111 1111 1111.
  Em hexadecimal, a gama é $\texttt{8000}_\texttt{H}$-$\texttt{9FFF}_\texttt{H}$.
  Trata-se igualmente de *descodificação total*.

O mapa de memória para o sistema da figura (a) é o seguinte:

Gama (hex)	Dispositivo
0000-7FFF	--
8000-9FFF	RAM2
A000-DFFF	--
E000-EFFF	RAM1
F000-FFFF	--

A análise do sistema da figura (b) faz-se de forma análoga.

- RAM1: $\text{CS} = A_{15} \cdot \overline{A_{13}} = 1$. Esta condição só é satisfeita se $A_{15} = 1, A_{13} = 0$.
  Logo, os endereços mapeadas na RAM1 têm o formato
  1?0X XXXX XXXX XXXX.
  O símbolo **?** indica que o bit correspondente não é usado na descodificação. Portanto, trata-se de *descodificação parcial*.
  Neste caso particular, dois endereços que difiram apenas no bit A_{14} têm o mesmo efeito em termos de acesso a memória: os dois endereços diferentes são mapeados no mesma posição física de memória. Temos, portanto, duas gamas de endereços equivalentes, que apenas diferem no valor de A_{14}. Para $A_{14} = 0$ a gama é:
  1000 0000 0000 0000 a 1001 1111 1111 1111.

Em hexadecimal, a gama é 8000_H-$9FFF_H$.

Para $A_{14} = 1$ a gama é:

1100 0000 0000 0000 a 1101 1111 1111 1111.

Em hexadecimal, a gama é $C000_H$-$DFFF_H$.

As duas gamas são mapeadas de forma *sobreposta* na RAM1. Por exemplo, os endereços 8000_H e $C000_H$ referem-se ambos à primeira posição física do circuito RAM1. A memória disponível não aumenta por ser usada descodificação parcial.

- O circuito RAM2 também é usado com descodificação parcial. Os endereços correspondentes têm o formato

  0?XX XXXX XXXX XXXX,

  a que correspondem as gamas 0000_H-$3FFF_H$ e 4000_H-$7FFF_H$.

O mapa de memória para o sistema da figura (b) é o seguinte:

Gama (hex)	Dispositivo
0000-3FFF	RAM2 (*)
4000-7FFF	RAM2 (*)
8000-9FFF	RAM1 (**)
A000-BFFF	--
C000-DFFF	RAM1 (**)
E000-FFFF	--

Os asteriscos assinalam gamas fisicamente sobrepostas.

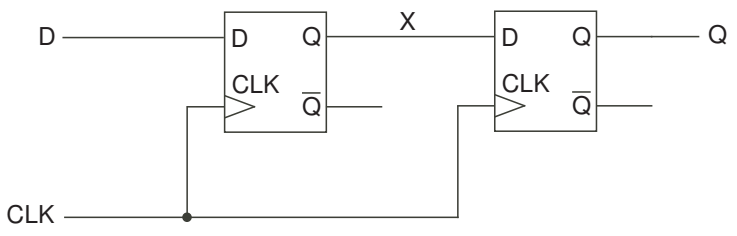
c) Para determinar os componentes usados basta consultar os mapas de memória obtidos na alínea anterior. Os resultados são os seguintes:

Endereço	Figura (a)	Figura (b)
B7FF	--	--
1000	--	RAM2
E7AA	RAM1	--
8000	RAM2	RAM1

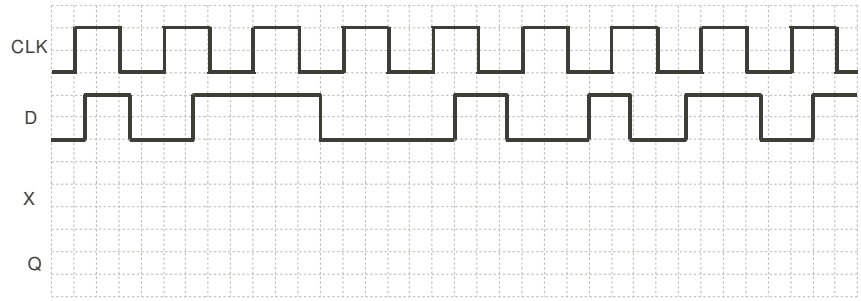
4.2 Exercícios propostos

Exercício 6

Assuma que no circuito seguinte os *flip-flops* do tipo D são sensíveis ao flanco ascendente do sinal de relógio e que inicialmente as saídas são nulas.

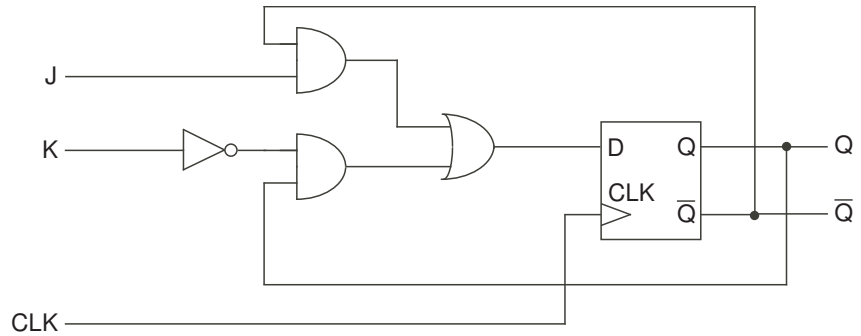


Represente a forma de onda da saída Q em resposta à entrada D representada na figura. Sugestão: comece por verificar qual o valor de X após cada transição do sinal de relógio.



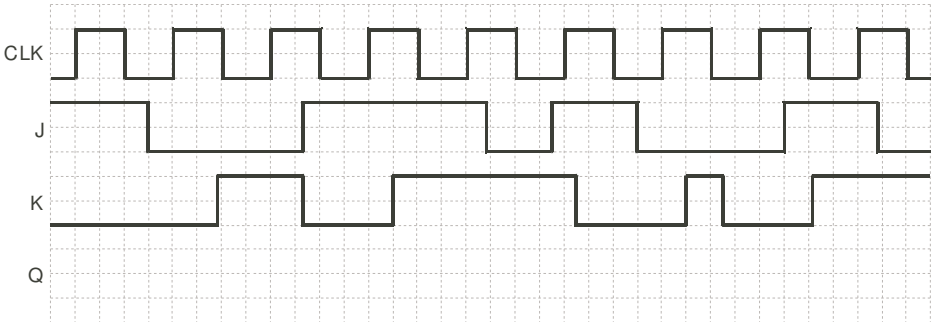
Exercício 7

Considere o seguinte circuito sequencial, constituído por portas lógicas e um *flip-flop* do tipo D, sensível ao flanco ascendente do sinal de relógio, em que no início $Q = 0$.



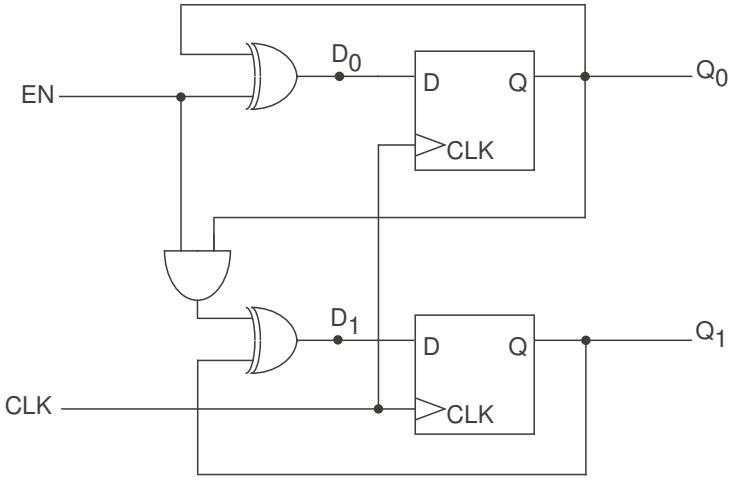
a) Indique a expressão da função lógica $D(J, K, Q)$ à entrada do *flip-flop*.

b) Considerando os valores das entradas J e K apresentados na figura seguinte, obtenha o valor da saída Q do circuito.



Exercício 8

Considere o circuito sequencial da figura seguinte.



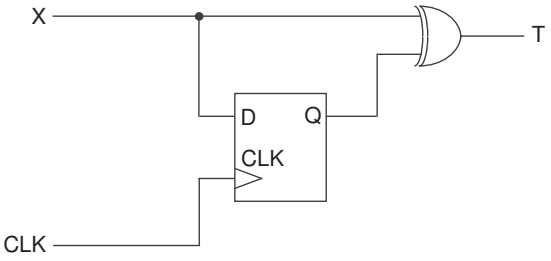
a) Escreva a expressão das entradas D_0 e D_1 dos *flip-flops*.

b) Represente as formas de onda correspondentes aos sinais Q_0 e Q_1, assumindo que o estado inicial dos *flip-flops* é "00" e que EN=1.

c) Mostre qual o estado do circuito após 4 transições consecutivas do sinal de relógio (CLK).

Exercício 9

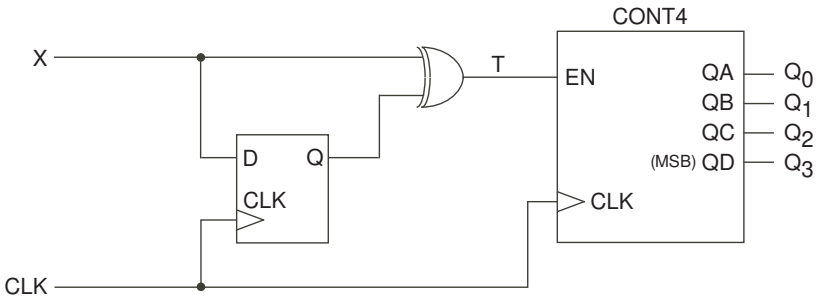
Considere o seguinte circuito, composto por um *flip-flop* e uma porta XOR (ou-exclusivo).



a) Assumindo que o estado inicial do *flip-flop* é 0, determine a sequência de valores na saída T se na entrada X ocorrer a sequência 0110001001111100 (um bit a cada ciclo do relógio).

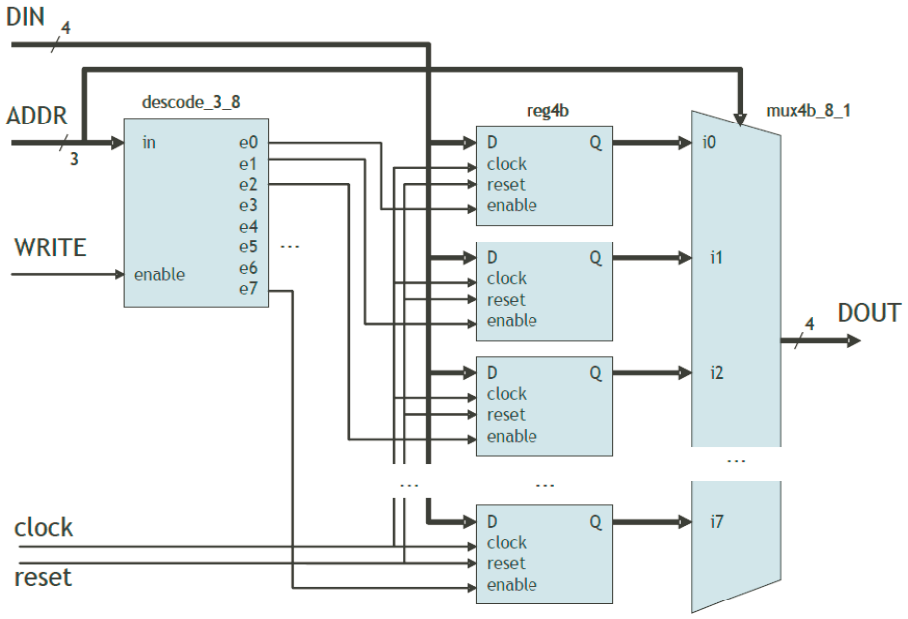
b) Descreva a função do circuito relacionando T com X.

c) Ao circuito anterior foi acrescentado um contador síncrono de 4 bits, tal como representado na figura seguinte. Identifique a relação entre as saídas $Q_3Q_2Q_1Q_0$ do contador e a entrada X do circuito.



Exercício 10

A figura mostra a constituição de um banco de registos.



Além dos 8 registos de 4 bits, fazem parte do circuito um descodificador de 3 para 8 e um multiplexador de 8 (conteúdos de 4 bits) para 1. A entrada DIN representa o valor de 4 bits a escrever num dos 8 registos, endereçado (identificado) por ADDR. A escrita ocorre se WRITE=1. A leitura de um registo é feita endereçando o registo pretendido, surgindo o valor na saída DOUT. As operações de escrita são síncronas com o sinal de relógio.

a) Expresse a capacidade de armazenamento do banco de registos em bytes.

b) Explique o que garante escrever um conteúdo num (e um só) determinado registo.

c) Admita que o conteúdo inicial dos registos é "0000" e que os valores das entradas são: DIN=1111, ADDR=001 e WRITE=1. Descreva que alterações ocorrem no circuito após uma transição do sinal de relógio.

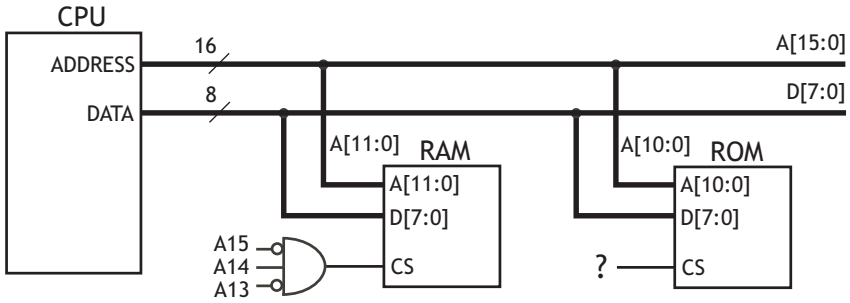
d) Explique a utilidade das entradas *enable* e *reset* dos registos.

e) Descreva a função desempenhada pelo descodificador de 3 para 8 e pelo multiplexador de 8 para 1, no contexto da sua utilização no banco de registos.

f) Mostre que alteração seria necessária efetuar para que fosse possível ler, simultaneamente, o conteúdo de dois registos diferentes, apresentando-o nas saídas DOUT1 e DOUT2.

Exercício 11

Um sistema de memória é constituído por uma memória RAM e uma memória ROM. O barramento de endereços possui 16 bits e o barramento de dados é de 8 bits. A figura seguinte mostra o correspondente diagrama de blocos, onde CS representa o sinal de *chip select* das memórias.



a) Determine o intervalo de endereços a que a RAM responde e justifique se a descodificação de endereços é total ou parcial.

b) Considere que a primeira posição da memória ROM tem o endereço 0xC800 e que o endereço de cada posição é único. Calcule o endereço da última posição da ROM.

c) Apresente o circuito de descodificação de endereços da ROM considerando as condições da alínea anterior.

Exercício 12

Um processador dispõe de um espaço de endereçamento de 64 KiB e de um barramento de dados com 8 linhas. O seu mapa de memória é o seguinte:

Gama	Dispositivo
0x0000-0x3FFF	ROM1
0x4000-0x7FFF	RAM1
0x8000-0xCFFF	-
0xD000-0xDFFF	RAM2
0xE000-0xFFFF	-

a) Determine as dimensões de cada um dos dispositivos de memória.

b) Determine as equações lógicas dos circuitos de descodificação de endereços (descodificação total).

c) Apresente o diagrama de blocos do sistema de memória com descodificação de endereços.

Exercício 13

Um sistema emprega um processador com 14 bits de endereço e 8 bits de dados. Assumir que se dispõe apenas de circuitos RAM de dimensão $2^{10} \times 4$ bits.

Pretende-se construir um subsistema de memória com 2048 posições (de 1 byte cada) a partir do endereço 0 (i.e., a gama de endereços vai de 0 a 2047). Endereços pares e ímpares devem ativar circuitos RAM diferentes. O sistema usa descodificação completa.

a) Apresente o esquema de ligações do sistema, incluindo o sub-sistema de descodificação de endereços.

b) O valor `0x9A` é guardado na posição 250. Onde fica guardada fisicamente a informação?

5 Desempenho

Alguns destes exercícios são extraídos ou adaptados do livro "Computer Organization and Design – The Hardware/Software Interface", D. Hennessy e J. Patterson, 4ª edição.

5.1 Exercícios resolvidos

Exercício 1
Um computador possui três classes de instruções, A, B e C. O CPI de cada uma delas é, respetivamente, 1, 2 e 3. Um projetista está a implementar um compilador e precisa de escolher uma de duas sequências de instruções a usar nesse computador. Dessas sequências é conhecido o número de instruções de cada classe, conforme mostra a tabela seguinte.

Sequência de instruções	Nº de instruções por classe		
	A	B	C
1	2	1	2
2	4	1	1

a) Determine em qual das duas sequências de instruções é executado o maior número de instruções.

b) Mostre qual das sequências é executada de forma mais rápida.

c) Calcule o valor do CPI para cada sequência.

a) Na sequência 1 são executadas $2 + 1 + 2 = 5$ instruções e na sequência 2 são executadas $4 + 1 + 1 = 6$ instruções. É pois na sequência 2 que são executadas mais instruções.

b) O número de ciclos é dado por

$$N_{\text{ciclos}} = \sum_i N_i \times \text{CPI}_i$$

onde N_i se refere ao número de instruções da classe i e CPI_i é o número de ciclos por instrução dessa classe.

Sequência 1: $N_{\text{ciclos}} = 2 \times 1 + 1 \times 2 + 2 \times 3 = 10$

Sequência 2: $N_{\text{ciclos}} = 4 \times 1 + 1 \times 2 + 1 \times 3 = 9$

A sequência 2 é a mais rápida, porque é executada em menos ciclos de relógio.

c)

$$\text{CPI} = \frac{N_{\text{ciclos}}}{N_{\text{inst}}}$$

Sequência 1: CPI = 10/5 = 2

Sequência 2: CPI = 9/6 = 1,5

Exercício 2

Considere um computador com três classes de instruções, A, B e C, para as quais o valor de CPI é 1, 2 e 3, respetivamente. Para ser executado nesse computador, um programa é obtido através de dois compiladores distintos, originando o número de instruções de cada classe indicado na tabela.

Sequência de instruções	Nº de instruções por classe		
	A	B	C
Compilador 1	5×10^9	1×10^9	1×10^9
Compilador 2	10×10^9	1×10^9	1×10^9

Assuma que o computador funciona a 500 MHz.

a) Indique a sequência que é executada em menos tempo.

b) Determine a qual das sequências de instruções corresponde o maior valor de MIPS (milhões de instruções por segundo).

a) O tempo de execução é dado por

$$t_{\text{exec}} = \frac{N_{\text{ciclos}}}{f_{\text{CLK}}}$$

sendo o número de ciclos de relógio determinado por

$$N_{\text{ciclos}} = \sum_i N_i \times \text{CPI}_i$$

Então, o tempo de execução da sequência de instruções obtida por cada compilador é

$$t_{\text{exec\_C1}} = \frac{(5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9}{500 \times 10^6} = \frac{10 \times 10^9}{500 \times 10^6} = 20\,\text{s}$$

e

$$t_{\text{exec\_C2}} = \frac{(10 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9}{500 \times 10^6} = \frac{15 \times 10^9}{500 \times 10^6} = 30\,\text{s}$$

Atendendo aos tempos de execução calculados, conclui-se que o compilador 1 gera o programa mais rápido.

b)

$$\text{MIPS} = \frac{N_{instr}}{t_{exec} \times 10^6}$$

Portanto,

$$\text{MIPS}_1 = \frac{(5+1+1)\times 10^9}{20\times 10^6} = 350$$

e

$$\text{MIPS}_2 = \frac{(10+1+1)\times 10^9}{30\times 10^6} = 400$$

Daqui se conclui que o compilador 2 proporciona uma taxa de execução de instruções superior.

Exercício 3
A execução de um programa num computador A, o qual funciona com um sinal de relógio de 2 GHz, demora 10 s. Um computador B que está a ser desenvolvido é capaz de executar o mesmo programa em 6 s. Este computador B pode atingir uma frequência de relógio superior à do computador A, embora consuma 1,2 vezes mais ciclos de relógio do que o computador A a executar o referido programa. Determine a frequência do sinal de relógio a que B pode funcionar.

O tempo de execução de um programa com N_{instr} instruções é dado por

$$t_{\text{exec}} = N_{\text{instr}} \times \text{CPI} \times T_{\text{CLK}}$$

A razão dos tempos de execução no computador A e no computador B é

$$\frac{t_{\text{exec\_A}}}{t_{\text{exec\_B}}} = \frac{N_{\text{instr}} \times \text{CPI}_A \times f_{\text{CLK\_B}}}{N_{\text{instr}} \times \text{CPI}_B \times f_{\text{CLK\_A}}} = \frac{10}{6}$$

Como $\text{CPI}_B = 1{,}2 \times \text{CPI}_A$, então resulta $f_{\text{CLK\_B}} = f_{\text{CLK\_A}} \times 1{,}2 \times \frac{10}{6} = 2 \times f_{\text{CLK\_A}}$, conclui-se que $f_{\text{CLK\_B}} = 4$ GHz.

Exercício 4
Considere dois computadores A e B que possuem o mesmo conjunto de instruções. O período do sinal de relógio dos computadores A e B é 250 ps e 500 ps, respetivamente, e o número de ciclos de relógio consumidos por instrução (CPI) é 2 no computador A e 1,2 no computador B, respetivamente. Mostre qual dos computadores é o mais rápido a executar um programa.

$$t_{exec_A} = N_{instr} \times 2 \times 250 = 500 \times N_{instr} \text{ ps}$$

e

$$t_{exec_B} = N_{instr} \times 1{,}2 \times 500 = 600 \times N_{instr} \text{ ps}$$

Então,

$$\frac{t_{exec_B}}{t_{exec_A}} = \frac{6}{5} = 1{,}2$$

O computador A é o mais rápido.

Exercício 5

Um dado programa é executado em dois computadores diferentes, A e B, tendo-se obtido os seguintes indicadores:

Indicador	A	B
Nº de instruções	10^7	8×10^6
Frequência	4 GHz	4 GHz
CPI	1	1,1

a) Indique qual o computador com maior valor de MIPS.

b) Indique qual o computador mais rápido.

a)

$$\text{MIPS} = \frac{N_{instr}}{t_{exec} \times 10^6} = \frac{N_{instr}}{N_{instr} \times \text{CPI} \times T_{CLK} \times 10^6} = \frac{f_{CLK}}{\text{CPI} \times 10^6}$$

Logo,

$$\text{MIPS}_A = \frac{4 \times 10^9}{1 \times 10^6} = 4 \times 10^3$$

e

$$\text{MIPS}_B = \frac{4 \times 10^9}{1{,}1 \times 10^6} \approx 3636$$

A é o computador com maior valor de MIPS.

b)

$$t_{exec} = N_{instr} \times \text{CPI} \times T_{CLK} = \frac{N_{instr}}{\text{MIPS} \times 10^6}$$

$$t_{exec_A} = \frac{10^7}{4 \times 10^3 \times 10^6} = 2{,}5\,\text{ms}$$

$$t_{exec_B} = \frac{8 \times 10^6}{3636 \times 10^6} \approx 2{,}2\,\text{ms}$$

B é o computador mais rápido.

Exercício 6

Para um dado programa foram obtidas as seguintes medidas:

- Ocorrência de operações de vírgula flutuante (VFL) (exceto raiz quadrada): 25 %
- Ocorrência da operação de raiz quadrada: 2 %
- CPI médio de operações em VFL: 4,0
- CPI de raiz quadrada: 20
- CPI médio para outras instruções: 1,33

Considere duas alternativas para melhorar o desempenho: reduzir o CPI da operação de raiz quadrada para 2 ou baixar o CPI de todas as instruções de VFL para 2,5. Compare as duas alternativas usando a equação de desempenho do CPU.

O número de instruções e a frequência do relógio do computador continuam a ser os mesmos nas duas alternativas. Basta por isso comparar os valores de CPI.

$$\text{CPI}_1 = 0{,}25 \times 4 + 0{,}02 \times 2 + (1 - 0{,}25 - 0{,}02) \times 1{,}33 \approx 2{,}01$$

$$\text{CPI}_2 = 0{,}25 \times 2{,}5 + 0{,}02 \times 20 + (1 - 0{,}25 - 0{,}02) \times 1{,}33 \approx 2{,}00$$

Embora os valores sejam muito próximos, na segunda alternativa ("... baixar o CPI de todas as instruções de VFL para 2,5") o tempo de execução é menor, ou seja, o desempenho é superior.

Exercício 7

Um processador executa um programa em que 30 % do tempo de execução é gasto em adições de vírgula flutuante, 25 % em multiplicações de vírgula flutuante e 10 % em divisões de vírgula flutuante. Para a nova geração desse processador, a equipa de projeto propõe três aperfeiçoamentos possíveis na unidade de vírgula flutuante, consistindo em tornar o:

1. somador duas vezes mais rápido;
2. multiplicador três vezes mais rápido;
3. divisor dez vezes mais rápido.

Indique qual destas alternativas poderá proporcionar o maior aumento de desempenho.

A lei de Amdahl pode ser expressa por

$$s_{total}(f) = \frac{1}{(1-f) + \frac{f}{s}}$$

onde f representa a fração de tempo gasto na execução da parte melhorada e s traduz o aumento de rapidez dessa parte.

A aplicação da lei de Amdahl às três situações descritas permite pois quantificar o aumento de desempenho:

1. $f = 0{,}3$ e $s = 2 \quad \rightarrow \quad s_{total}(f) = \frac{1}{0{,}7+0{,}3/2} = 1{,}18$
2. $f = 0{,}25$ e $s = 3 \quad \rightarrow \quad s_{total}(f) = \frac{1}{0{,}75+0{,}25/3} = 1{,}20$
3. $f = 0{,}1$ e $s = 10 \quad \rightarrow \quad s_{total}(f) = \frac{1}{0{,}9+0{,}1/10} = 1{,}10$

O maior ganho de desempenho é obtido com a melhoria do multiplicador de vírgula flutuante (alternativa 2). Pode ainda concluir-se que tornar o divisor muito mais rápido leva ao menor aumento de desempenho devido à baixa ocorrência de divisões em vírgula flutuante (10 %). Na verdade, mesmo tornando o divisor infinitamente mais rápido, o *speed-up* obtido seria 1,11.

Exercício 8
A execução de um programa num computador demora 1000 s, sendo 75 % do tempo gasto em multiplicações e divisões. Pretende-se remodelar o computador com *hardware* mais rápido para a realização das operações de multiplicação e divisão. Calcule quanto mais rápidas devem ser as operações de multiplicação e divisão para que o programa seja:

a) três vezes mais rápido; b) quatro vezes mais rápido.

a) Aplicando a lei de Amdahl, o aumento de rapidez do programa é dado por

$$s_{total}(f) = \frac{1}{(1-f) + \dfrac{f}{s}}$$

onde f representa a fração de tempo gasto na execução da parte melhorada e s traduz o aumento de rapidez dessa parte. Assim,

$$3 = \frac{1}{(1-0{,}75) + \dfrac{0{,}75}{s}} = \frac{1}{\dfrac{1}{4} + \dfrac{3}{4 \times s}} = \frac{4 \times s}{s+3}$$

resultando

$$3 \times s + 9 = 4 \times s \Leftrightarrow s = 9$$

isto é, as operações devem ser 9 vezes mais rápidas.

b) Da mesma forma, resulta

$$4 = \frac{1}{(1-0{,}75) + \dfrac{0{,}75}{s}} = \frac{1}{\dfrac{1}{4} + \dfrac{3}{4 \times s}} = \frac{4 \times s}{s+3}$$

e

$$4 \times s + 12 = 4 \times s$$

que não tem solução, concluindo-se ser impossível o programa ficar 4 vezes mais rápido.

5.2 Exercícios propostos

Exercício 9
A tabela apresentada mostra a frequência de funcionamento de 3 processadores, o número de instruções executadas em cada um deles e o respetivo tempo de execução.

Processador	F_{CLK} (GHz)	Nº de instruções	Tempo (s)
P_1	2	20×10^9	7
P_2	1,5	30×10^9	10
P_3	3	90×10^9	9

a) Calcule o número de instruções executadas por ciclo de relógio, por cada processador.

b) Calcule a frequência a que o processador P_2 deve funcionar para que o tempo de execução das instruções seja igual ao apresentado para P_1.

c) Calcule o número de instruções a executar em P_2 de modo a que o seu tempo de execução seja igual ao de P_3.

Exercício 10

Considere que dois processadores distintos, P_1 e P_2, implementam o mesmo conjunto de instruções e que estas se podem enquadrar em quatro classes diferentes (A, B, C e D). A frequência e o CPI por classe de instruções de cada implementação encontram-se na tabela.

Processador	F_{CLK} (GHz)	CPI A	CPI B	CPI C	CPI D
P_1	1,5	1	2	3	4
P_2	2	2	2	2	2

a) Seja um programa com 1 milhão de instruções divididas pelas quatro classes da seguinte forma: 10 % são da classe A, 20 % são da classe B, 50 % são da classe C e 20 % são da classe D. Verifique qual dos processadores é mais rápido a executar o programa.

b) Calcule o CPI médio de cada processador.

c) Calcule o número de ciclos de relógio necessários à execução do programa em cada processador.

Exercício 11

Dois processadores, P_1 e P_2, implementam de forma diferente o mesmo conjunto de instruções, composto por cinco classes de instruções. P_1 funciona com um relógio de 4 GHz e P_2 funciona a 6 GHz. O número médio de ciclos de relógio para cada classe de instruções é dado pela tabela que se segue.

Classe	CPI de P_1	CPI de P_2
A	1	2
B	2	2
C	3	2
D	4	4
E	3	4

O desempenho de pico *(peak performance)* é definido como a cadência máxima a que um computador pode executar a sequência de instruções mais favorável (para uma dada tarefa). Determinar o desempenho de pico de P_1 e P_2 expresso em instruções por segundo. Comentar o resultado.

Exercício 12

A tabela seguinte apresenta o número de operações em vírgula flutuante (VFL) executadas em três programas diferentes e o respetivo tempo de execução em três computadores A, B e C, distintos.

Programa	Nº operações VFL	Tempo de execução (s)		
		Comp. A	Comp. B	Comp. C
1	5×10^9	2	5	10
2	20×10^9	20	20	20
3	40×10^9	200	50	15

a) Mostre qual o computador mais rápido em termos do tempo total de execução.

b) Determine quantas vezes esse computador é mais rápido do que os outros dois.

Exercício 13

A tabela mostra o número total de instruções executadas por um programa, bem como a sua distribuição por tipo.

Total	Aritméticas	*Store*	*Load*	Salto
700	500	50	100	50

a) Assuma que o tempo de execução das instruções aritméticas corresponde a 1 ciclo de relógio, *loads* e *stores* correspondem a 5 ciclos de relógio e as instruções de salto correspondem a 2 ciclos de relógio. Calcule o tempo de execução do programa num processador a 2 GHz.

b) Calcule o valor do CPI para o programa.

c) Se o número de instruções de *load* for reduzido para metade, quanto mais rápida é a execução do programa e qual o valor do CPI?

Exercício 14

Um processador executa um programa constituído por vários tipos de instruções, entre as quais existem instruções de transferência de dados. Supondo que é possível melhorar o tempo de execução das instruções de transferência de dados em 11 vezes, calcule a percentagem do tempo de execução gasto por essas instruções de forma a conseguir-se melhorar o desempenho do processador em 5 vezes.

Exercício 15

Medindo o tempo de acesso a disco verificou-se que 80 % do tempo de execução de um determinado programa num computador era gasto em acessos ao disco. Substituindo o disco de tal computador constatou-se que este era duas vezes mais rápido.

a) Calcule quanto melhorou o desempenho do computador.

b) Verifique qual a melhoria de desempenho que seria alcançada se o disco tivesse um tempo de acesso desprezável.

Exercício 16

Suponha que se pretende melhorar um computador acrescentando-lhe uma unidade vetorial. (Estas unidades implementam instruções específicas para o tratamento de vetores.) Um cálculo executado em modo vetorial é 10 vezes mais rápido que o normal.

a) Desenhe um gráfico que mostre o ganho de rapidez (*speedup*) em função da percentagem de tempo de execução em modo vetorial (percentagem de vetorização).

b) Que percentagem de vetorização é necessária para se obter um ganho de rapidez de 2?

c) Considerar o caso da alínea anterior. Com a utilização da unidade vetorial, o computador demora metade de tempo a realizar os cálculos. Em que percentagem desse novo tempo (mais curto) é que a unidade vetorial está a ser usada?

d) Que percentagem de vetorização é necessária para obter metade do máximo ganho de rapidez possível?

e) Suponha que se determinou empiricamente que a percentagem de vetorização é de 70 %. Os projetistas de *hardware* acham que é possível duplicar o desempenho da unidade vetorial. Em alternativa, o grupo de compilação poderia tentar aumentar a utilização do modo vetorial, como forma de aumentar o desempenho. Que aumento da percentagem de vetorização (em relação à situação atual) seria necessário para obter o mesmo resultado que a duplicação do desempenho de *hardware*? Que alternativa é mais aconselhável?

Exercício 17

Um processador suporta 4 classes de instruções. Durante a execução de um dado programa, a percentagem de tempo gasta com instruções de cada classe foi o seguinte:

Classe	A	B	C	D
Tempo (%)	30	20	40	10

Uma nova versão do processador executa instruções de classe A três vezes mais rapidamente e instruções de classe C quatro vezes mais rapidamente. Determinar o ganho de rapidez *(speedup)* obtido pela nova versão.

6 Linguagem *assembly*

6.1 Exercícios resolvidos

Exercício 1
Para as seguintes expressões aritméticas (números inteiros de 64 bits), especifique um mapeamento de variáveis para registos e o fragmento de código *assembly* ARM que as implementa.

a) `f = g - (f + 5)`

b) `f = A[12] + 17`

O primeiro passo neste tipo de problemas é escolher uma atribuição de variáveis a registos. Cada variável é atribuída a um registo. Como a arquitetura ARM possui 31 registos de uso geral, trata-se de uma tarefa simples porque, neste caso, se pode usar um registo diferente para cada variável.

a) Uma possível atribuição de registos a variáveis é a seguinte (escolha arbitrária):

`X0`: `f` `X1`: `g`

O fragmento de código que realiza os cálculos desejados é:

```
add     X0, X0, 5               // Calcula f = f + 5
sub     X0, X1, X0              // Calcula f = g - f
```

Após esta sequência de duas instruções, `X0` contém o novo valor associado a `f`. O cálculo da primeira parte da expressão (instrução `add`) pode guardar o resultado intermédio no registo `X0`, porque a segunda instrução estabelece o valor final correto.

b) Possível atribuição de variáveis a registos:

`X0`: `f` `X6`: endereço base de `A`

Como cada elemento de uma sequência de inteiros tem 8 bytes, o elemento de índice 12 da sequência `A` está guardado a partir da posição de memória cujo endereço é dado por:

$$\texttt{endereço base de A} + 12 \times 8$$

A primeira instrução deve ir buscar o valor guardado nessa posição.

```
ldur    X0, [X6, #96]           // Carrega valor da posição X6+96
add     X0, X0, #17             // Soma-lhe o valor 17
```

Exercício 2

Assuma as seguintes condições iniciais:

`X0 = 0x00000000BEADFEED` `X1 = 0x00000000DEADFADE`

a) Determine o valor de `X2` após a execução da seguinte sequência de instruções:

```
lsl     X2, X0, 4
orr     X2, X2, X1
```

b) Determine o valor de `X2` após a execução da seguinte sequência de instruções:

```
lsr     X2, X0, 3
and     X2, X2, 0x00000000FFFFFFEF
```

Em binário, os valores iniciais dos registos são:

$$\texttt{X0} = 0\ldots0\ 1011\ 1110\ 1010\ 1101\ 1111\ 1110\ 1110\ 1101_2$$
$$\texttt{X1} = 0\ldots0\ 1101\ 1110\ 1010\ 1101\ 1111\ 1010\ 1101\ 1110_2$$

a) A primeira instrução desloca o valor de `X0` quatro bits para a esquerda. Nos 4 bits menos significativos são introduzidos zeros. O resultado da operação é guardado em `X2`; o registo `X0` fica inalterado. O valor de `X2` depois da execução da primeira instrução é:

$$\texttt{X2} = 0\ldots0\ 1011\ 1110\ 1010\ 1101\ 1111\ 1110\ 1110\ 1101\ 0000_2$$

A instrução `orr` calcula a função ou-inclusivo de cada bit de `X2` com o bit de `X0` situado na mesma posição. O resultado é guardado em `X2`. O resultado da operação `orr` é 1 sempre que pelo menos um dos operandos seja 1. Logo:

$$\texttt{X2} = 0\ldots0\ 1011\ 1111\ 1110\ 1111\ 1111\ 1111\ 1110\ 1101\ 1110_2 = \texttt{0x0000000DFEFFFEDE}$$

b) A instrução `lsr` desloca o valor de `X0` três posições para a direita, introduzindo zeros pela esquerda; os 3 bits menos significativos perdem-se. O valor de `X2` depois da execução da primeira instrução é:

$$\texttt{X2} = 000\ 0\ 0\ldots0\ 0001\ 0111\ 1101\ 0101\ 1011\ 1111\ 1101\ 1101_2$$

A instrução `and` calcula a função e-lógico de cada bit de `X2` com o bit correspondente da constante `0x00000000FFFFFFEF`

$$\texttt{00000000FFFFFFEF}_{16} = 0\ldots0\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 1111_2$$

O resultado é guardado em `X2`. O e-lógico de dois bits tem resultado 1 apenas se ambos os operandos forem 1. Neste caso, os operandos são dados pelo conteúdo de `X2` e pela constante indicada. O valor final de `X2` é:

$$\texttt{X2} = 0\ldots0\ 0001\ 0111\ 1101\ 0101\ 1011\ 1111\ 1100\ 1101_2 = \texttt{0x0000000017D5BFCD}$$

Exercício 3

Assuma as seguintes condições iniciais:

$$\texttt{X0} = \text{FFFF FFFF FFFF FFFF}_{16}$$

$$\texttt{X1} = 0\ldots0\ 0011\ 1111\ 1111\ 1000\ 0000\ 0000\ 0000\ 0000_2$$

Determine o valor de `X1` após a execução do fragmento seguinte:

```
        cmp     X0, X1
        bge     ELSE
        b       DONE
ELSE:   add     X1, X1, 2
DONE:   ...
```

A primeira instrução compara os conteúdos de `X0` e `X1` alterando o valor das *flags* (registo NZVC) de acordo com o resultado da comparação. A operação realizada é equivalente a:

```
X0 - X1
```

Neste caso os valores dos indicadores (*flags)* são alterados para N=1, Z=0, C=1 e V=0.

O salto condicional (segunda instrução) é tomado se as *flags* N e V apresentarem o mesmo valor, o que se verifica quando `X0` é maior ou igual a `X1` (a condição `ge` interpreta os valores dos registos como sendo números em complemento para 2). Como neste caso o conteúdo de `X0` é negativo e o conteúdo de `X1` é positivo, o salto não será tomado.

Em consequência, a terceira instrução a ser executada é a de salto incondicional (a instrução `b`). Esta instrução leva o fluxo de execução a passar para o fim do fragmento apresentado (etiqueta `DONE`). A instrução `add` não é executada.

Como o conteúdo de `X1` não foi alterado, o seu valor são sofre alteração.

Exercício 4

Apresenta-se a seguir um programa em *assembly*. Na quinta linha do programa é invocada uma sub-rotina, através da instrução `bl` (*branch with link*), designada por `par`. Esta sub-rotina, cujo código não é fornecido, tem como objetivo determinar se um número é par. Admita que a sub-rotina recebe esse número no registo `X0` e que devolve no registo `X0` o valor 1 caso seja par, ou 0 no caso contrário.

```
            mov   X10, 0x00000000FF000000
            mov   X1, 3
            mov   X2, 0
loop:       stur  X0, [X10]
            bl    par               // Chama rotina par
            cmp   X0, 0
            beq   step
            add   X2, X2, 1
step:       ldur  X0, [X10]
            add   X0, X0, 1
            adds  X1, X1, -1
            bne   loop
```

a) Considerando que antes da execução do programa o registo `X0` possui o valor 8, indique o conteúdo dos registos `X0` e `X2` após a execução do programa.

b) Tendo em consideração a descrição que foi realizada, implemente a rotina `par`.

a) Em algumas situações é útil preservar o valor do registo `X0` (onde, segundo a convenção as sub-rotinas devem retornar o resultado), para isso uma solução possível é armazenar o conteúdo do registo numa posição de memória antes da invocação da sub-rotina e voltar a carregar esse valor após a execução da mesma. Neste caso foi utilizado o endereço de memória 0x00000000FF000000 para armazenar o conteúdo do registo `X0`.

O programa realiza 3 iterações, incrementando em cada uma delas o valor `X0`, que inicialmente é 8. Em cada iteração é determinada a paridade do valor em `X0` invocando a rotina `par`. Caso o valor em `X0` seja par o registo `X2` é incrementado. No final da execução do programa `X2`=2, correspondendo aos 2 números pares encontrados (8 e 10), e `X0`=11, correspondendo ao resultado da adição de uma unidade em cada iteração ao valor de `X0`.

b) O bit menos significativo de um número par é 0. A rotina seguinte baseia-se nesta propriedade.

```
par:        add   X0, X0, 1
            and   X0, X0, 1
            ret
```

Exercício 5

a) Escreva um fragmento de código *assembly* que determina se um dado número inteiro `N` está presente numa sequência `SEQ`. Assuma a seguinte atribuição de registos:

- `N`→`X0`
- endereço-base de `SEQ`→`X1`
- número de elementos de `SEQ`→`X2`
- resultado→`X0`

b) Converta o fragmento anterior numa sub-rotina chamada `pesq` (de "pesquisa"). Os argumentos da função seguem a ordem indicada na alínea anterior.

c) Use a sub-rotina anterior num fragmento que determina quantos elementos de uma sequência `SEQ1` estão presentes noutra sequência `SEQ2`. Usar a seguinte atribuição de registos:

- endereço-base de `SEQ1`→`X7`
- número de elementos de `SEQ1`→`X8`
- endereço-base de `SEQ2`→`X9`
- número de elementos de `SEQ2`→`X10`
- resultado→`X5`

a) O fragmento consiste num ciclo em que se varia o registo `X1` de forma a conter o endereço de elementos sucessivos de `SEQ`. O número de iterações é, no máximo, igual ao número de elementos de `SEQ`, que é decrementado de uma unidade em cada iteração.

O ciclo pode terminar, quando se encontra um elemento igual ao procurado, nesse caso é colocado em `X3` o valor 1 e a instrução de salto para a etiqueta fim é executada.

Se o valor procurado não existir em `SEQ`, o ciclo é terminado porque o contador de elementos vem a 0. Neste caso, o valor de `X3` não é alterado, mantendo o valor inicial estabelecido na primeira instrução.

Na última linha o valor de `X3` é transferido para o registo `X0` de forma a deixar o resultado em `X0` tal como é pedido no enunciado.

```
1          mov     X3, 0      // resultado a Zero
2  prox:   cmp     X2, 0      // terminar?
3          beq     fim
4          ldur    x4, [x1]    // obter elemento
5          cmp     x4, X0      // elemento = N?
6          bne     seg
7          mov     X3, 1      // encontrado, resultado a 1
8          b       fim
9  seg:    add     x1, x1, 8 // atualizar endereço
10         sub     x2, x2, 1 // ajustar numero de elementos
11         b       prox
12 fim:    mov     x0, x3      // colocar resultado em X0
```

b) Para transformar o fragmento numa sub-rotina é necessário alterá-lo para corresponder às convenções de invocação: os argumentos nos registos `X0`–`X7` e o resultado nos registos `X0` e `X1`.

A atribuição de registos passa a ser a seguinte:

- `N`→`X0`
- endereço-base de `SEQ`→`X1`
- número de elementos de `SEQ`→`X2`
- resultado→`X0`

```
1  pesq:   mov     X3, 0       // resultado a Zero
2  L1:     cmp     X2, 0       // terminar?
3          beq     L3
4          ldur    X4, [X1]     // obter elemento
5          cmp     X4, X0       // elemento = N?
6          bne     L2
7          mov     X3, 1       // encontrado, resultado a 1
8          b       L3
9  L2:     add     X1, X1, 8   // atualizar endereço
10         sub     X2, X2, 1   // ajustar numero de elementos
11         b       L1
12 L3:     mov     X0, X3       // colocar resultado em X0
13         ret
```

A ultima instrução da sub-rotina (`ret`), tem com função fazer com que no final da execução da sub-rotina o programa retorne para o programa principal continuando a executar as suas instruções normalmente.

c) O fragmento consiste num ciclo em que se "percorre" a sequência `SEQ1`. As linhas 2–3 verificam se existem elementos a processar. Em caso afirmativo, obtém-se o próximo elemento de memória; as linhas 9–10 procedem à atualização do contador de elementos e do endereço do próximo elemento.

A sub-rotina `pesq` é usada para procurar um dado elemento de `SEQ1` em `SEQ2`. As linhas 4–6 preparam a invocação da sub-rotina, colocando os argumentos nos registos apropriados (valor a procurar em `X0`, endereço-base de `SEQ2` em `X1` e número de elementos de `SEQ2` em `X2`).

A linha 8 processa o resultado da invocação (registo `X0`). Se o valor foi encontrado em `SEQ2`, o contador `X5` é incrementado de uma unidade.

```
1           mov     X5, 0       // inicializar contador
2   ciclo:  cmp     X8, 0       // mais elementos de SEQ1?
3           beq     stop         // terminar
4           ldur    X0, [X7]     // obter um elemento de SEQ1
5           mov     X1, X9       // onde pesquisar
6           mov     X2, X10      // nº de elementos a pesquisar
7           bl      pesq         // invocar sub-rotina
8           add     X5, X5, X0   // atualizar contador
9           add     X7, X7, 8   // próximo endereço
10          sub     X8, X8, 1   // decrementar nº de elementos
11          b       ciclo        // repetir
12  stop:   ...
```

6.2 Exercícios propostos

Exercício 6

Para as seguintes expressões aritméticas (números inteiros de 64 bits), especifique um mapeamento de variáveis para registos e o fragmento de código *assembly* ARMv8 que as implementa.

a) `f = g + ( j + 2)`

b) `k = a + b - f + d - 30`

c) `f = g + h + A[4]`

d) `f = g - A[B[10]]`

e) `f = k - g + A[h + 9]`

f) `f = g - A[B[2] + 4]`

Exercício 7

Para os seguintes fragmentos de código *assembly* ARMv8, indique um mapeamento entre registos e variáveis e determine as expressões simbólicas correspondentes.

a)
```
add     x0,x0,x1
add     x0,x0,x2
add     x0,x0,x3
add     x0,x0,x4
```

b)
```
ldur    x0,[x6, 8]
```

c) Assumir que `X6` contém o endereço-base da sequência `A[]`.
```
add     x6, x6, -40
lsl     x10, x1, 3
add     x6, x6, x10
ldur    x0, [x6, 16]
```

Exercício 8

Assuma as seguintes condições iniciais:

```
X0 = 0x5555555555555555          X1 = 0x0123456789ABCDEF
```

Determine o valor de `X2` após a execução das sequências de instruções seguintes.

a)
```
lsl     x2, x0, 4
orr     x2, x2, x1
```

b)
```
lsl     x2, x0, 4
and     x2, x2, -1
```

c)
```
lsr     x2, x0, 3
and     x2, x2, 0x00EF
```

Exercício 9

Os processadores RISC como o ARM implementam apenas instruções muito simples. Este exercício aborda exemplos de hipotéticas instruções mais complexas.

a) Considere uma instrução hipotética `abs` que coloca num registo o valor absoluto de outro registo.

`abs  X2, X1` é equivalente a `X2 ← |X1|`

Apresente uma sequência de instruções ARMv8 que realiza esta operação.

b) Repita a alínea anterior para a instrução `sgt`, em que `sgt  X1, X2, X3` é equivalente a `se X2 > X3 então X1 ← 1 senão X1 ← 0`.

Exercício 10

Considere o seguinte fragmento de código *assembly* ARMv8:

```
L1:     stur  X4,[X5]
        lsl   X4,X4,4
        add   X5,X5,8
        cmp   X4, 0
        b.ne  L1
```

Assuma os seguintes valores iniciais:

```
X4 = 0x12345678          X5 = 0x7D0
```

Explique como é alterada a memória durante a execução do fragmento de código. Apresente numa tabela o endereço e o conteúdo das posições de memória alteradas pela execução do fragmento de código.

Exercício 11

Numa zona de memória (endereço-base em `X4`) está uma sequência de números inteiros (de 64 bits) diferentes de 0. A sequência é terminada por um zero.

Escreva um fragmento de código *assembly* que determina o número de valores da sequência. O valor final, 0, não entra para a contagem. O resultado deve ser guardado em `X0`.

Exercício 12

Considerar os seguintes fragmentos de código *assembly*.

Fragmento 1:
```
LOOP:   cmp     X1, 0
        b.eq    DONE
        add     X2, X2, 2
        add     X1, X1, -1
        b       LOOP
DONE:           ...
```

Fragmento 2:
```
LOOP:   mov     X3, 0xA
LOOP2:  add     X2, X2, 2
        adds    X3, X3, -1
        b.ne    LOOP2
        adds    X1, X1, -1
        b.ne    LOOP
DONE:   ...
```

a) Assumir a seguinte situação inicial: `X1=10` e `X2=0`. Determinar, para cada fragmento, o valor final de `X2`.

b) Assuma agora que `X1=N` (com `N>0`). Determinar, para cada fragmento, o número de instruções executadas em função de `N`.

Exercício 13

Escrever um fragmento para determinar se duas sequências de números inteiros (64 bits) têm o mesmo conteúdo. As sequências têm 100 elementos. Usar a seguinte atribuição de registos:

`endereço-base da sequência A` → `X4`
`endereço-base da sequência B` → `X5`

O resultado, a guardar em `X0`, deve ser 1, se as duas sequências tiverem o mesmo conteúdo ou 0 no caso contrário.

Exercício 14

Escreva um fragmento de código *assembly* ARMv8 para determinar quantos números ímpares estão contidos numa sequência de 50 números inteiros (de 64 bits). Assuma que o endereço-base da sequência está contido no registo `X0`. O resultado deve ficar no registo `X7`.

Exercício 15

Pretende-se escrever um programa que permita realizar diversas tarefas envolvendo sequências de números inteiros (64 bits) em memória. Para que o programa resulte estruturado e o código seja facilmente reutilizado, o seu desenvolvimento deve basear-se na chamada de sub-rotinas, realizando cada uma destas sub-rotinas uma tarefa específica.

Escreva um programa que realiza as tarefas a seguir descritas usando uma sub-rotina para cada tarefa.

Nota: deve gerir a utilização de registos por forma a respeitar as convenções de chamada e de retorno das sub-rotinas.

a) Somar todos os elementos de uma sequência.

b) Determinar o número de elementos ímpares da sequência.

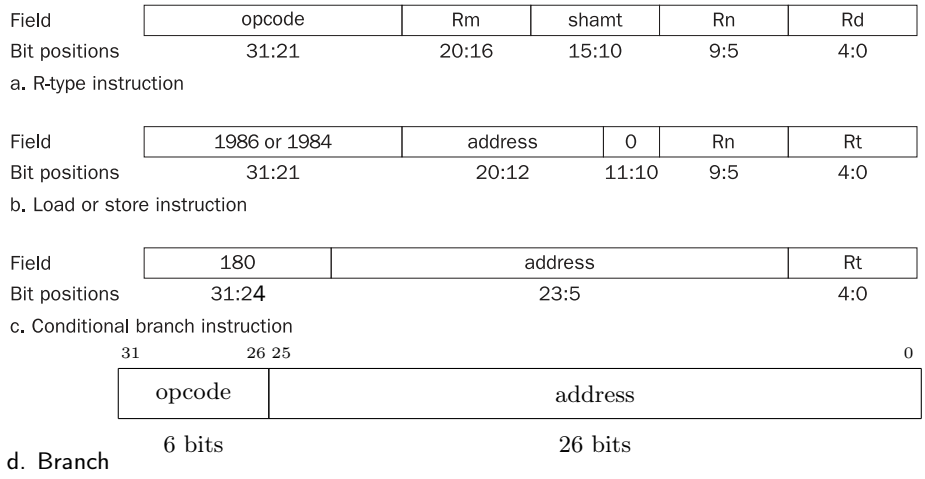
c) Determinar o número de elementos que são iguais ou superiores a um valor dado.

d) Determinar se duas sequências com o mesmo comprimento são iguais.

7 Organização de um CPU

As figuras foram extraídas do livro "Computer Organization and Design – ARM Edition", Patterson & Hennessey.

Resumo dos formatos de codificação e opcodes



Instrução	Opcode
ADD	100 0101 1000
SUB	110 0101 1000
AND	100 0101 0000
ORR	101 0101 0000
LDUR	111 1100 0010
STUR	111 1100 0000
CBZ	101 1010 0
B	000 101

ALU trabalha em 3 contextos diferentes.

1. instruções lógico-aritméticas: `ALUOp[1:0]=10`
2. cálculo de endereços: `ALUOp[1:0]=00`
3. comparação: `ALUOp[1:0]=01`

7.1 Exercícios resolvidos

Exercício 1

Considere o CPU da figura 7.1. Determine o valor de todos os sinais de controlo durante a execução da seguinte instrução:

```
ADD    X3, X8, X11
```

Indique também quais os componentes com funções úteis.

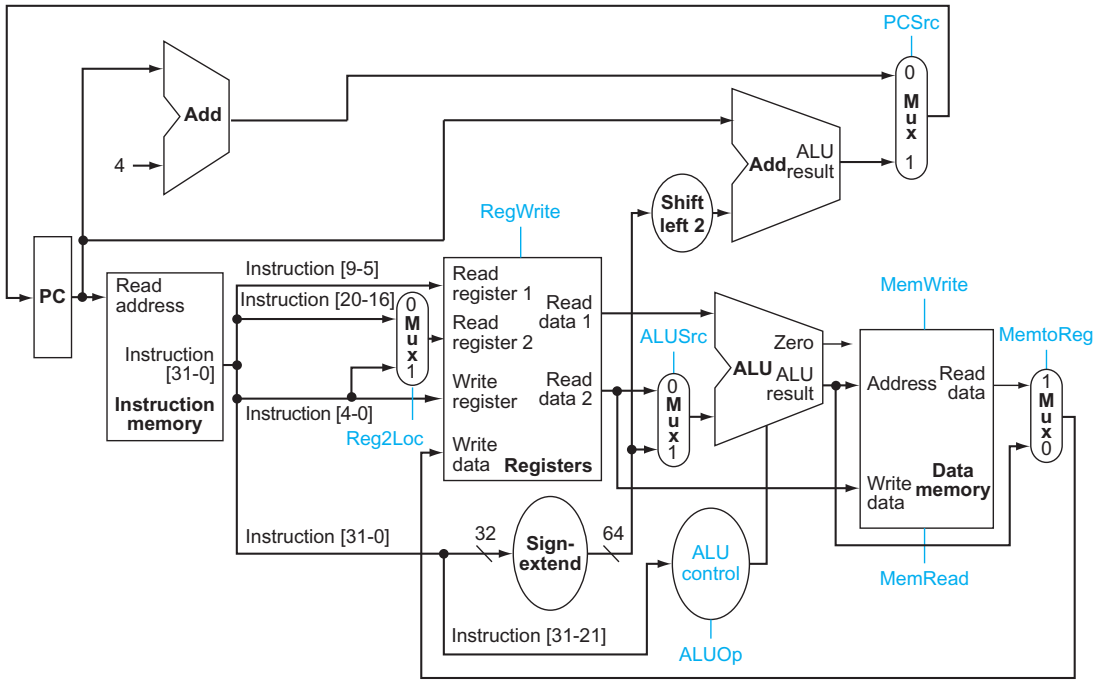


Figura 7.1: CPU com multiplexadores e sinais de controlo

Os valores dos sinais de controlo são determinados pela operação efetuada, que estabelece a forma como os recursos do CPU são usados.

A instrução `ADD` usa dois registos como fonte de operandos e um como destino do resultado. Como a instrução altera o banco de registos (para guardar o resultado da soma), deve ter-se `RegWrite=1`. Como o valor que é guardado no registo provém da ALU e não da memória, tem-se `MemtoReg=0`. Nas instruções tipo R, a especificação do registo do segundo operando é feita pelos bits 20:16 da instrução, pelo que `Reg2Loc=0`.

Esta instrução não acede a memória, seja para leitura, seja para escrita. Por isso, `MemRead=0` e `MemWrite=0`. Como também não é um salto condicional, `Branch=0`, o que implica `PCSrc=0` (porque `PcSrc = AND(Zero, Branch)`).

A instrução `ADD` usa a ALU para realizar a operação (adição) sobre dois operandos provenientes de registos. Um operando da ALU provém sempre do banco de registos; a origem do segundo é determinada pelo sinal `ALUSrc`. Como neste caso o segundo operando também provém do banco de registos, tem-se `ALUSrc=0`. A operação da ALU é controlada pelo sinal de 2 bits `ALUOp`. Como se trata de uma instrução do tipo R, tem-se `ALUOp=`10_2. Esta instrução não afeta o fluxo de instruções, pelo que `PCSrc=0`.

Todos os módulos têm funções úteis, exceto a memória da dados `(Data memory)`, o módulo de extensão de sinal `(Sign extend)`, o módulo de deslocamento `(Shift left 2)` e o somador usado para calcular o destino dos saltos relativos (`Add` ligado a `Shift left 2`).

Exercício 2

Suponha que a unidade de controlo usada com o CPU da figura 7.1 possui um defeito de fabrico que faz com que um determinado sinal de controlo tenha o valor fixo 0. Indique, justificando, quais as instruções que deixam de funcionar corretamente, considerando que o defeito ocorre em:

a) `RegWrite` b) `ALUOp[1]`

Para que uma instrução não funcione como esperado é necessário que o sinal, em operação correta, assuma um valor diferente daquele que é imposto pelo defeito de fabrico.

a) Como o defeito de fabrico impõe sempre `RegWrite=0`, todas as instruções que requerem `RegWrite=1` funcionarão incorretamente. As instruções com `RegWrite=1` são as que alteram o conteúdo de um registo: as instruções do tipo R e a instrução `LDUR`.

b) O sinal `ALUOp[1]=1` somente para as instruções lógico-aritméticas. Como o defeito impõe `ALUOp[1]=0`, a ALU comporta-se nestas instruções como se estivesse a efetuar um cálculo de endereços, i.e., a efetuar uma adição. Logo, o defeito de fabrico impede todas as instruções lógico-aritméticas, exceto a adição, de funcionarem corretamente: `SUB`, `AND` e `ORR`.

Exercício 3

Processadores podem apresentar defeitos de fabrico. Suponha que se pretende detetar se um CPU tem um certo defeito procedendo da seguinte forma:

1. preencher PC, registos e memória de instruções com valores selecionados apropriadamente;
2. executar *uma única* instrução;
3. ler os valores de PC, registos e memória de dados.

Os valores lidos são examinados para determinar se uma dada falta está presente ou não.

a) Apresente um teste que permita determinar se o bit 12 da saída da memória de instruções está sempre a 0 (*stuck-at-0*).

b) Repita a alínea anterior para uma falta *stuck-at-1* (saída sempre a 1). É possível usar um único teste para os dois casos? Se sim, como; se não, porquê?

Para detetar um defeito é preciso provocar uma situação em que o sinal afetado assuma normalmente o valor complementar daquele que é imposto pelo defeito.

a) Este defeito afeta o bit 12 das instruções obtidas de memória. Nas instruções de tipo R, este bit pertence ao campo `shamt`, que não é usado. Por isso, é necessário usar uma instrução de outro tipo. Uma solução possível:

```
CBZ X31, 128
```

Na ausência de defeito, PC ← PC + 128 × 4. Na presença do defeito, $128_{10} = 000\ldots10000000_2$ transforma-se em $000\ldots00000000_2$, já que o bit 12 (o décimo terceiro bit da instrução) é forçado a 0. Portanto, o valor de PC mantém-se.

b) Por um raciocínio análogo ao da alínea anterior, é necessário usar uma constante com o bit 12 a zero. Por exemplo: `CBZ X31, 256`.

Na ausência de defeito, temos PC ← PC + 256 × 4; no caso contrário, toma o valor PC ← PC + 384 × 4 (o campo de endereço muda de $000\ldots100000000_2$ para $000\ldots110000000_2$)

Não é possível testar simultaneamente a presença de um ou outro defeito, porque isso exigiria usar uma única instrução para impor condições opostas no local do defeito.

Exercício 4

A tabela seguinte indica a latência máxima dos blocos usados na implementação do CPU conforme indicado na figura 7.1.

I-Mem	Add	Mux	ALU	Regs	D-Mem	Controlo
400 ps	100 ps	30 ps	120 ps	200 ps	350 ps	100 ps

A latência dos restantes módulos é nula.
Determine o caminho crítico (caminho de propagação de sinais com maior latência) para a instrução `AND`.

É preciso encontrar o caminho ao longo do qual a propagação de sinais demora mais. O tempo de propagação deve ser contado a partir da saída do registo `PC` (que muda no início de cada ciclo). Para cada instrução, existem duas tarefas a realizar:

1. Calcular o endereço da próxima instrução.
2. Realizar a operação da instrução corrente.

Estas duas tarefas são independentes, exceto no caso das instruções que alteram o fluxo de instruções (saltos condicionais e incondicionais). Para a instrução `AND` as tarefas são independentes.

Para calcular o tempo de propagação associado a cada módulo, é preciso determinar o respetivo sinal de entrada que é atualizado em último lugar. Para um módulo de latência L com N entradas *relevantes* cujos valores estejam atualizados nos instantes $t_1, t_2, \ldots, t_n$, e latência máxima L, a saída estará atualizada no instante $t = \max(t_1, t_2, \ldots, t_n) + L$.

Por exemplo, a saída da memória de instruções `(I-MEM)` está atualizada 400 ps após o início da instrução. A unidade de controlo `(Control)` atualiza todos os sinais de controlo no instante

$$400\,\text{ps} + 100\,\text{ps} = 500\,\text{ps}$$

Para determinar o caminho crítico, é preciso determinar qual das duas tarefas demora mais.

1ª tarefa: A propagação de dados relativos ao cálculo do próximo endereço passa pelos seguintes módulos:

`Add` → `Mux`

A saída de `Add` está atualizada após 100 ps.

As entradas relevantes de `Mux` são a saída do 1º somador e o sinal de controlo. Este último está atualizado no instante $t = 500\,\text{ps}$, conforme calculado anteriormente. Portanto, a saída de `Mux` está atualizada em

$$t = \max(100\,\text{ps}, 500\,\text{ps}) + 30\,\text{ps} = 530\,\text{ps}$$

Portanto, o endereço da próxima instrução está calculado após 530 ps.

2ª tarefa: A realização da operação utiliza os seguintes módulos:

$$\texttt{I-Mem} \rightarrow \texttt{Control} \rightarrow \texttt{Mux} \rightarrow \texttt{Regs} \rightarrow \texttt{Mux} \rightarrow \texttt{ALU} \rightarrow \texttt{Mux}$$

O 1º `Mux` é o que está ligado à entrada `Read Register 2`, o 2º `Mux` é o da entrada da ALU e o 3º é o que está à saída da memória `D-Mem`. É de notar que a entrada de seleção do 1º `Mux` só está disponível ao fim de $400\,\text{ps} + 100\,\text{ps} = 500\,\text{ps}$. Por este motivo, nesta instrução (e em outras do tipo R) a unidade de controlo pertence ao caminho crítico.

A saída superior de `Regs` está pronta no instante

$$t = 400\,\text{ps} + 200\,\text{ps} = 600\,\text{ps}$$

A saída inferior de `Regs` está pronta no instante

$$t = 400\,\text{ps} + 100\,\text{ps} + 30\,\text{ps} + 200\,\text{ps} = 730\,\text{ps}$$

No instante $t = 730\,\text{ps}$ já todos os sinais de controlo estão atualizados, o que implica que, daqui para a frente, estes já *não* influenciam o cálculo do tempo de propagação.

Portanto, o resultado da operação está pronto (i.e., disponível na entrada `Write data` do banco de registos) no instante

$$t = 730\,\text{ps} + 30\,\text{ps} + 120\,\text{ps} + 30\,\text{ps} = 910\,\text{ps}$$

Falta verificar se as demais entradas relevantes do módulo `Regs` estão atualizadas neste instante. A entrada `RegWrite` está atualizada desde o instante 500 ps, porque é um sinal de controlo. A entrada `Write register` está pronta desde o instante 400 ps. Portanto, a entrada `Write data` é a última a ficar atualizada.

Concluindo, o caminho crítico é $\texttt{I-Mem} \rightarrow \texttt{Control} \rightarrow \texttt{Mux} \rightarrow \texttt{Regs} \rightarrow \texttt{Mux} \rightarrow \texttt{ALU} \rightarrow \texttt{Mux}$, com um tempo de propagação de 910 ps.

Exercício 5

Assuma que o processador da figura 7.1 recebe a seguinte instrução:

`1111 1000 0100 0001 0000 0000 0100 0011`

a) Indique os valores das saídas dos componentes `Sign-extend` e `Shift-left2`.

b) Indique os valores de entrada da unidade de controlo da ALU.

c) Qual é o valor de PC depois da execução da instrução?

Assuma agora que o conteúdo dos registos é o seguinte:

X0	X1	X2	X3	X4	X5	X6	X7	X12
0	1	4	3	-4	5	6	8	1

d) Indique os valores de saída de cada multiplexador.

e) Indique os valores de entrada da ALU e dos dois somadores.

f) Indique os valores de todas as entradas do banco de registos.

A resolução usa a figura 7.1 como referência.

a) O componente `Sign-extend` tem à entrada a instrução, e produz um valor de 64 (com sinal) a partir de um campo da instrução. Neste caso, `Instr[26]=0`, pelo que o campo escolhido é `Instr[20:12]`. O valor do campo é 000010000_2, logo o valor à saída de `Sign-Extend` é:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 00001\ 0000_2$$

ou `0x0000000000000010`

O componente `Shift-left2` produz à saída o valor da entrada deslocado de 2 bits para a esquerda: `0x000000000040`.

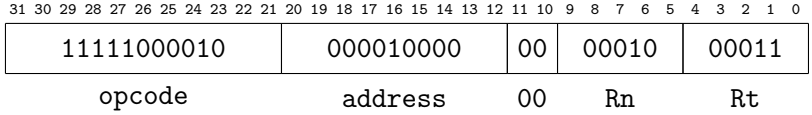
b) Para responder às alíneas seguintes é necessário saber de que instrução se trata (para se conhecerem os valores dos sinais de controlo).

Consultado a lista de *opcodes*, temos $\texttt{opcode=11111000010}_2 = 1986_{10}$, logo trata-se de uma instrução `LDUR`.

Para a instrução `LDUR`, a ALU é usada para calcular o endereço dos dados fazendo uma adição: `ALUop=00`.

c) Como a instrução `LDUR` não altera o fluxo de instruções, `PC` $\leftarrow$ `PC + 4`.

d) Descodificando a instrução, tem-se:



Portanto, `Rn=2` e `Rt=3`. A instrução completa é: `LDUR X3,[X2,#16]`.

Como a instrução usa a ALU para calcular a soma `X2+16`, o multiplexador à entrada da ALU seleciona a constante (após extensão de sinal): a saída desse multiplexador tem o valor 16.

Como na instrução `LDUR` o registo de destino é definido por `Instr[0:4]`, o multiplexador que alimenta a entrada `Read register 2` pode selecionar qualquer valor uma vez que é irrelevante para esta instrução. Logo, a sua saída pode ser 00010_2=2 ou 00011_2=3.

A instrução `LDUR` transfere um valor de memória para um registo. Portanto, a saída do multiplexador controlado por `MemtoReg` tem o valor proveniente da posição de memória lida (valor armazenado em memória a partir da posição 20, pois `X2`=4).

Esta instrução não altera o fluxo de instruções. Portanto, a saída do multiplexador controlado por `PCSrc` é igual a `PC+4`.

e) A entrada superior da ALU tem o valor do endereço-base (registo `X2`):4.

A entrada inferior da ALU tem a constante 16 (em 64 bits).

As entradas do somador da esquerda são: `PC` e 4.

As entradas do somador de destino de saltos são: `PC+4` e o valor da constante multiplicado por 4 (efeito do deslocamento de 2 bits para a esquerda) $16 \times 4 = 64$ (em 64 bits).

f) Os valores das entradas do banco de registos são:

- `Read register 1`: valor de `Rn`: 2.
- `Read register 2`: o valor de `Reg2Loc` é irrelevante para esta instrução. Logo, o valor de `Read register 2` pode ser 00010_2=2 ou 00011_2=3.
- `Write register`: valor de `Rt`: 3
- `Write data`: valor lido de memória (do endereço 20).
- `RegWrite`: 1 (sinal de controlo para habilitar o armazenamento do novo valor).

7.2 Exercícios propostos

Exercício 6

Para as seguintes instruções em código-máquina ARMv8, determine as instruções *assembly* correspondentes. Indique o tipo de codificação, bem como os valores de todos os campos.

a) `0xF81F80A0`

b) `0xF8404023`

Exercício 7

Considere o CPU da figura 7.1. Determine o valor de todos os sinais de controlo durante a execução das instruções indicadas a seguir. Indique também quais os componentes com funções úteis.

a) `LDUR X3,[X7,#64]`

b) `STUR X8,[X2,#128]`

c) `CBZ X9,#-5`

Exercício 8

Suponha que a unidade de controlo usada com o CPU da figura 7.1 possui um defeito de fabrico que faz com que um determinado sinal de controlo tenha o valor fixo 0. Indique, justificando, quais as instruções que deixam de funcionar corretamente, considerando que a falha ocorre em:

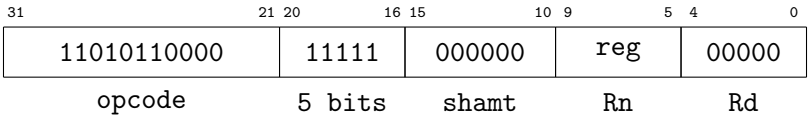
a) `PCSrc`

b) `MemWrite`

Exercício 9

Pretende-se implementar a instrução `BR reg` *(branch register)*. Esta instrução passa o fluxo de execução para a instrução cujo endereço está no registo `reg`: `PC` ← `reg`.

A codificação de `BR` tem o formato R com alguns campos fixos:



Indique quais os elementos e sinais de controlo que devem ser acrescentados ao CPU indicado na figura 7.1. Defina os valores dos sinais de controlo para a nova situação.

Exercício 10

Considere a instrução `LDUR X8, [X17,#40]`. Para responder às questões seguintes refira-se à figura 7.1.

a) Represente a instrução em linguagem-máquina.

b) Qual é o número de registo fornecido à entrada `Read register 1`? O valor deste registo é usado?

c) Responda à mesma questão para a entrada `Read register 2`.

d) Qual é o número de registo fornecido à entrada `Write register`? O valor deste registo é realmente alterado?

e) Qual é o valor dos sinais de controlo `MemRead` e `MemWrite`?

f) Responda às alíneas anteriores para a instrução:

```
Label CBZ X11, Label
```

Exercício 11

A tabela seguinte indica a latência máxima dos blocos usados na implementação do CPU indicado na figura 7.1.

I-Mem	Add	Mux	ALU	Regs	D-Mem	Controlo
400 ps	100 ps	30 ps	120 ps	200 ps	350 ps	100 ps

Determine o caminho crítico (caminho de propagação de sinais com maior latência) para as instruções seguintes, bem como o tempo de propagação correspondente.

a) `LDUR`

b) `CBZ`

Exercício 12

Pretende-se projetar a unidade de controlo para uma implementação do CPU em que os componentes têm a seguinte latência máxima:

I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-extend	Shift-left2	ALU Ctrl
400 ps	100 ps	30 ps	120 ps	200 ps	350 ps	20 ps	0 ps	50 ps

a) Determine o tempo disponível para a unidade de controlo gerar o sinal `MemWrite` sem aumentar a latência da implementação.

b) Determine qual é o sinal de controlo que pode demorar mais a ser gerado e qual é o tempo de que a unidade de controlo dispõe para esse efeito sem aumentar a latência da implementação.

c) Determine qual é o sinal de controlo que pode demorar *menos* a ser gerado (sinal crítico) e qual é o tempo de que a unidade de controlo dispõe para esse efeito sem aumentar a latência da implementação.

8 Memória Cache

8.1 Exercícios resolvidos

Exercício 1
Considere uma memória *cache* do tipo *direct-mapped*, com 8 blocos e 1 palavra por bloco. Complete a seguinte tabela, que representa a evolução do estado da memória *cache* para os dez acessos consecutivos indicados. Inicialmente, a memória *cache* está vazia. Os endereços são endereços de *palavras*.
Apresente também o conteúdo da memória *cache* após o último acesso.

Acesso	Endereço	Hit/Miss	Conteúdo do bloco	
			inicial	final
1	22			
2	26			
3	22			
4	26			
5	16			
6	3			
7	16			
8	8			
9	27			
10	10			

Como a memória tem 8 blocos, o índice do bloco é definido pelo resto da divisão do endereço por 8.

O efeito da sequência de acessos é descrito a seguir. A operação de obtenção do resto é representada por %:

$$a \ \% \ b = \text{resto da divisão de } a \text{ por } b.$$

1. Endereço 22 →posição 22 % 8 = 6; falha *(miss)*; cache[6] ←mem[22].
2. Endereço 26 →posição 26 % 8 = 2; falha *(miss)*; cache[2] ←mem[26].
3. Endereço 22 →posição 22 % 8 = 6; acerto *(hit)*; cache[6] tem conteúdo de mem[22].
4. Endereço 26 →posição 26 % 8 = 2; acerto *(hit)*; cache[2] tem conteúdo de mem[26].
5. Endereço 16 →posição 16 % 8 = 0; falha *(miss)*; cache[0] ←mem[16].

6. Endereço 3 →posição 3 % 8 = 3; falha *(miss)*; cache[3] ←mem[3].
7. Endereço 16 →posição 16 % 8 = 0; acerto *(hit)*; cache[0] tem conteúdo de mem[16].
8. Endereço 8 →posição 8 % 8 = 0; falha *(miss)*; cache[0] tinha conteúdo de mem[16]; cache[0] ←mem[8].
9. Endereço 27 →posição 27 % 8 = 3; falha *(miss)*; cache[3] tinha conteúdo de mem[3]; cache[3] ←mem[27].
10. Endereço 10 →posição 10 % 8 = 2; falha *(miss)*; cache[2] tinha conteúdo de mem[26]; cache[2] ←mem[10].

Portanto, a tabela fica:

Acesso	Endereço	Hit/Miss	Conteúdo do bloco	
			inicial	final
1	22	M	-	mem[22]
2	26	M	-	mem[26]
3	22	H	mem[22]	mem[22]
4	26	H	mem[26]	mem[26]
5	16	M	-	mem[16]
6	3	M	-	mem[3]
7	16	H	mem[16]	mem[16]
8	8	M	mem[16]	mem[8]
9	27	M	mem[3]	mem[27]
10	10	M	mem[26]	mem[10]

O conteúdo final da memória *cache* é:

Bloco	Conteúdo
0	mem[8]
1	-
2	mem[10]
3	mem[27]
4	-
5	-
6	mem[22]
7	-

Exercício 2

Um processador funciona a 2 GHz e possui uma memória *cache* unificada. A taxa de falhas no acesso à cache é 9% e o tempo de acesso à memória principal é 50 ns.

a) Calcule a penalidade de falha em ciclos.

b) Medições efetuadas para um conjunto de *benchmarks* revelaram que o número de ciclos de protelamento em acessos a memória (por instrução) é $\mathrm{CPI}_{\mathrm{prot}} = 12$.

Determine a percentagem de instruções que acede a dados.

a) Para obter a penalidade de falha em ciclos, determina-se o número de ciclos que o processador executa durante o tempo de acesso a memória principal.

$$T = \frac{1}{F} = 0{,}5\,\text{ns} \qquad \text{logo} \qquad p_f = \frac{50}{0{,}5} = 100$$

b) Seja n_a o número de acessos a memória por instrução e t_f a taxa de falhas.

O número médio de ciclos gastos em protelamento devidos a falhas no acesso a *cache* é:

$$\text{CPI}_{\text{prot}} = t_f \times p_f \times n_a$$

Portanto:

$$n_a = \frac{\text{CPI}_{\text{prot}}}{t_f \times p_f},$$

o que implica

$$n_a = \frac{12}{0{,}09 \times 100} = \frac{4}{3}$$

Cada instrução requer obrigatoriamente um acesso para a sua obtenção. Acessos adicionais são necessariamente acessos a dados. Então, o número médio de acessos a dados é: $n_a - 1 = \frac{1}{3} = 33{,}3\,\%$.

8.2 Exercícios propostos

Alguns exercícios foram extraídos ou adaptados do livro "Computer Organization and Design – The Hardware/Software Interface", Hennessy & Patterson, 4ª edição.

Exercício 3

Um CPU tem endereços de 18 bits e uma memória *cache* de mapeamento direto para palavras (32 bits). A memória *cache* tem 16 posições. A política de escrita é *write-through*. O conteúdo inicial da memória *cache* está indicado na tabela (em hexadecimal).

	conteúdo	etiqueta	v
0	12345678	abc	1
1	6548feab	123	1
2	3c1f56fd	678	1
3	afd12498	567	0
4	6198fa34	b7c	1
5	1929aaaa	8d1	0
6	bbabeedd	cd3	1
7	1123aa56	456	1
8	7611432a	001	1
9	ffffeffe	877	1
10	ddedd556	777	0
11	4444cccc	198	1
12	7627abed	fdf	1
13	8768888a	479	1
14	71672912	655	0
15	22256733	111	1

a) Determine o número de bits da etiqueta e do índice. Qual é o espaço de endereçamento? Qual a quantidade máxima de memória usável num sistema baseado neste CPU?

b) Indique as alterações da memória *cache* para a seguinte sequência de acessos (L=leitura, E=escrita):

tipo	endereço	valor
L	2df10	-
L	23454	-
L	3f7b0	-
E	048c4	1212abab
E	3f7d0	00001111
E	1dde8	aaaabbbb

Exercício 4

Assuma que o CPU do problema anterior é usado com uma memória *cache* também semelhante à anterior, mas que usa a política de escrita *write-back*. O conteúdo inicial da memória *cache* está indicado na tabela (em hexadecimal).

	conteúdo	etiqueta	v	d
0	12345678	abc	1	1
1	6548feab	123	0	0
2	3c1f56fd	678	1	0
3	afd12498	567	0	0
4	6198fa34	b7c	1	0
5	1929aaaa	8d1	0	1
6	bbabeedd	cd3	1	1
7	1123aa56	456	1	0
8	7611432a	001	1	1
9	ffffeffe	877	1	1
10	ddedd556	777	0	0
11	4444cccc	198	1	1
12	7627abed	fdf	0	1
13	8768888a	479	1	0
14	71672912	655	0	0
15	22256733	111	1	0

a) Explique a finalidade do campo `d`.

b) Indique as alterações da memória *cache* para a seguinte sequência de acessos (L=leitura, E=escrita):

tipo	endereço	valor
L	2df10	-
E	2df10	33334444
E	10c64	9999aaaa
L	23454	-
L	3f7b0	-
E	21de4	bbbb7777
E	2afdc	1212abab
E	3f7b0	00001111

Exercício 5

Uma memória *cache* do tipo *write-through* com 4 blocos de 8 bytes é usada como *D-cache* num processador ARMv8.

O conteúdo da memória *cache* é o seguinte (em hexadecimal):

bloco	conteúdo								etiqueta	v
	7	6	5	4	3	2	1	0		
0	2a	5d	04	aa	78	00	9c	23	2900002	1
1	1b	b2	34	bb	7a	10	9f	a3	0000893	1
2	99	52	36	cc	88	b0	3c	2b	7bcd001	1
3	42	15	90	cd	71	ab	3f	6d	65abfff	0

Assuma que `X0` = `0x52000044` e `X1`=`0xf79a0034`.

a) Qual é o comprimento da etiqueta?

b) Qual é o valor lido de memória por uma operação de 32 bits que use o endereço guardado em `X0`?

c) Que alterações da memória *cache* são causadas pela escrita de um valor de 4 bytes (*doubleword*) da posição especificada por `[X0, -4]`?

d) Qual é o valor obtido por uma instrução de leitura de 1 byte do endereço da posição de memória especificada pelo conteúdo de `X1`?

Exercício 6

Para cada um dos sistemas indicados a seguir considere que o tempo de acesso a memória principal é de 70 ns e que 36% das instruções acedem a dados em memória. O acesso a memória principal tem início após falta de acesso à memória *cache*.

Processador	Tamanho	Taxa de faltas	Tempo de acerto
P1	1 KiB	11,4%	0,62 ns
P2	2 KiB	8,0%	0,66 ns

a) Assumindo que é o tempo de acerto que determina o período de relógio, determine as frequências de operação dos dois sistemas.

b) Determine o tempo médio de acesso à memória para os dois casos.

c) Assumindo um CPI básico de 1, qual é o CPI de cada um dos processadores? Qual é o processador mais rápido?

Exercício 7

Um CPU (com F=1 GHz) está equipado com memórias *cache* para instruções e para dados, cujas taxas de faltas são, respetivamente, 5 % e 10 %. O tempo de acesso a memória principal é 80 ns (a acrescentar ao tempo de acesso a memória *cache*).

Em média, 40 % das instruções de um programa acedem a dados (i.e., são *load* ou *store*).

a) Determine a taxa de faltas global da memória *cache* em número de faltas por 1000 instruções.

b) Suponha que se pretendia equipar o CPU com uma memória *cache* unificada. Determine a máxima taxa de faltas desta alternativa para que ela apresente o mesmo desempenho que a versão *split cache*.

c) Assuma que, na ausência de faltas de *cache*, o CPU tem CPI_{ideal}=1,2. Determine o CPI efetivo para os seguintes casos:

   i) sistema sem memória *cache*;

   ii) sistema com memória *cache*.

Soluções dos exercícios propostos

1 Aritmética binária

Exercício 9

a) 100000000_2; 100_H

b) 11111111111_2; $7FF_H$

c) $11000{,}01_2$; $18{,}4_H$

d) A parte fracionária não tem representação finita. Usando 4 bits para a parte fracionária, a solução é: $100{,}0011_2; 4{,}3_H$.

e) $16_{10}; 10_H$

f) $4{,}125_{10}; 4{,}2_H$

g) $11110_2; 30_{10}$

h) $43981_{10}; 1010101111001101_2$

i) $171{,}75_{10}; 10101011{,}11_2$

j) 456_H

Exercício 10

a) 1001110_2

b) $1010{,}001_2$

c) 1001_2

d) 10101_2

e) 1111010_2

f) $11000100{,}01_2$

g) 101100_2

Exercício 11

a) $3 = 0011_2$; $2 = 0010_2$; $-3 = 1011_2$

b) $3 + 2 = 0101_2$; $2 + (-3) = 1001_2$

c) Impossível: 14 não é representável.

Exercício 12

Sinal e grandeza: $[-7; 7]$; complemento para 2: $[-8; 7]$

Exercício 13

a) SG: 00010010_2; C2: 00010010_2

b) SG: 00110001_2; C2: 00110001_2

c) SG: 10110001_2; C2: 11001111_2

d) SG: 10000011_2; C2: 11111101_2

e) SG: 11100100_2; C2: 10011100_2

f) SG: 01110011_2; C2: 01110011_2

g) SG: 11111111_2; C2: 10000001_2

h) SG: impossível; C2: 10000000_2

Exercício 14

a) 1010100_2

b) Não ocorre *overflow*; a diferença de números com o mesmo sinal é sempre representável.

Exercício 15

a) 10000001_2; resultado correto (não ocorre *overflow*).

b) 01011111_2; resultado errado (ocorre *overflow*).

c) 00000000_2; resultado correto (não ocorre *overflow*).

Exercício 16

a) X SS: 227; C2: -29
Y SS: 72; C2: 72

b) SS: $\underline{1}00101011$; resultado errado (ocorre *overflow*).
C2: 00101011; resultado correto (não ocorre *overflow*).

Exercício 17

a) $P_{\mathrm{H}} =$7A; $Q_{10} =$34

b) i) 0011100_2; resultado errado, pois a soma necessita de 8 bits (há *overflow*).
ii) 0011100_2; resultado correto, a soma é representável (não há *overflow*).

Exercício 18

a) $A + B = 1000100_2 = 68_{10}$.

b) $A + B = 0100000_2 = 32_{10}$.

c) $A + B = 0100100_2 = 36_{10}$.

Exercício 19

a) 01001010_2

b) -92

c) $M - N = 10100110_2$; resultado errado (ocorre *overflow*).
$N - M = 01011010_2$; resultado errado (ocorre *overflow*).

Exercício 20

a) $S = -56$; $T = 17$

b) $S = 10111000_2$; $T = 00010001_2$

c) $S + T = 10100111_2$; resultado correto (não há *overflow* ao somar as grandezas).

Exercício 21

a) $Y = 49_{\text{H}}$; $Z = 10011110_2$

b) 01011010_2; ocorre *overflow*, i.e., o resultado está errado (a soma de números negativos não pode ser positiva).

c) 01000011_2

Exercício 22

a) $x \in [0; 31]$

b) 192

c) -64

d) 9 bits

e) $192 : 011000000_2$
$-64 : 111000000_2$

2 Vírgula flutuante

Exercício 7

a) $1{,}01100000000000000000000_2$

b) 00000100_2

c) $-22{,}0$

Exercício 8

a) $41FA0000_{\text{H}}$

b) $BF200000_{\text{H}}$

c) 00000000_{H}

d) 44805000_{H}

Exercício 9

a) $(0\ 00000001\ 00000000000000000000000)_2 \rightarrow 1{,}17549435 \times 10^{-38}$

b) $(0\ 11111110\ 11111111111111111111111)_2 \rightarrow 3{,}4028235 \times 10^{38}$

c) $(1\ 00000001\ 00000000000000000000000)_2 \rightarrow -1{,}17549435 \times 10^{-38}$

d) $(1\ 11111110\ 11111111111111111111111)_2 \rightarrow -3{,}4028235 \times 10^{38}$

Exercício 10

a) Número negativo. O expoente é $1 - 3 = -2$, resultando em $-2^{-2} \times 1{,}1101_2$, ou seja $-2^0 \times 0{,}011101_2$.

Em decimal, corresponde a $-(2^{-2} + 2^{-3} + 2^{-4} + 2^{-6}) = -2^{-6} \times (2^4 + 2^3 + 2^2 + 1) = -\dfrac{16+8+4+1}{64} = -\dfrac{29}{64}$.

b) Número positivo. O expoente é $5 - 3 = 2$, resultando em $2^2 \times 1{,}1001_2$, ou seja $2^0 \times 110{,}01_2$.

Em decimal, corresponde a $6 + 2^{-2} = 6{,}25$.

c) Número positivo. $\dfrac{3}{8} = \dfrac{1+2}{8} = 2^{-3} \times 11_2 = 2^{-2} \times 1{,}1_2$.

O expoente da representação é $-2 + 3 = 1 = 001_2$ e o significando é $0{,}1000_2$.

A representação completa é $(0\ 001\ 1000)_2$.

Exercício 11

a) A: 42040000_{H}; B: $\text{C}0380000_{\text{H}}$

b) i) $A + B$: $41\text{F}10000_{\text{H}}$

ii) $B - A$: $\text{C}20\text{F}8000_{\text{H}}$

iii) $3 \times B$: $\text{C}10\text{A}0000_{\text{H}}$

c) $A + B = 30{,}125$ $B - A = -35{,}875$ $3 \times B = -8{,}625$.

Exercício 12

a) 40000000_{H} b) 42150000_{H} c) $\text{C}2250000_{\text{H}}$ d) $\text{C}29\text{D}0000_{\text{H}}$

Exercício 13

a) $01000001001110100000000000000000_2$ b) 41460000_{H} (falta indicar os passos)

Exercício 14

a) $11000001001001000000000000000000_2$ b) $13{,}25_{10}$ (falta indicar os passos)

Exercício 15

a) $\text{C}1400000_{\text{H}}$ b) $\text{C}2080000_{\text{H}}$ (falta mostrar os passos).

Exercício 16

Opção A.

Exercício 17

a) $3{,}3895314 \times 10^{38}$

b) $1{,}17549435 \times 10^{-38}$

c) representação de $2{,}6 \times 10^{6}$: `0x4a1e`
representação de $3{,}1 \times 10^{4}$: `0x46f2`
resultado da multiplicação: `0x5195` $\rightarrow$ 79993765888.

O resultado exato seria 80600000000. Logo, o erro absoluto é 606234112 e o erro relativo é 0,752 %. A representação mais curta provoca um erro, mas o respetivo valor relativo é pequeno. A implementação física de `Bfloat16` exige menos recursos que a de precisão simples e pode ser feita mais rapidamente.

3 Circuitos combinatórios

Exercício 10

a) $F(A, B, C, D, E) = A \cdot B + \overline{C} \cdot E$

b) $F(A, B, C) = \overline{A} \cdot \overline{C}$

c) $G(A, B, C) = A \cdot \overline{B} + A \cdot \overline{C}$

d) $F(A, B, C, D) = \overline{B} \cdot C + A \cdot C \cdot \overline{D}$

e) $F(W, X, Y, Z) = W + X \cdot \overline{Z} + X \cdot \overline{Y}$

f) $F(A, B, C, D) = \overline{A} \cdot C + C \cdot D$

Exercício 11

a)

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

b)

X	Y	Z	G
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

c)

W	X	Y	Z	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Exercício 12

a)

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

b) $F = (X + Z) \cdot (\overline{X} + \overline{Y} + \overline{Z})$. c) —

Exercício 13

a) $Y = A + B$. b) —

Exercício 14

a) $F = \overline{A} \cdot B + \overline{C}$

b)

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

c) Nota: usar a expressão simplificada.

Exercício 15

a)

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

$F = (X + Y + Z) \cdot \overline{X}$

b)

A	B	C	D	G
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$G = A \cdot B + \overline{A} \cdot \overline{B} \cdot C \cdot \overline{D} + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} + \overline{A} \cdot B \cdot C \cdot \overline{D} + A \cdot \overline{B} \cdot C \cdot \overline{D}$

Exercício 16

Nota: construir tabela de verdade, exprimir G na forma de produto de somas e simplificar.
$G = (A + B) \cdot (A + C) \cdot (\overline{A} + \overline{B} + \overline{C})$

Exercício 17

a)

A	B	S	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

b) $F(A, B, S) = A \cdot \overline{S} + B \cdot S$.

c) Nota: desenhar circuito lógico que realiza $F(A, B, S)$.

d) $G(X, Y, S) = X \cdot Y + X \cdot S + Y \cdot S$.

e) Nota: deduzir expressão da função realizada pelo circuito e simplificar. Alternativamente, pode ser construída a tabela de verdade a partir da expressão deduzida do circuito e da expressão obtida na alínea anterior, com fins comparativos.

Exercício 18

a)

A_3	A_2	A_1	A_0	S
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

b) $S = A_3 \cdot \overline{A_1} \cdot \overline{A_0}$

Exercício 19

a)

A	B	Ci	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

b) $S(A, B, Ci) = \overline{A} \cdot \overline{B} \cdot Ci + \overline{A} \cdot B \cdot \overline{Ci} + A \cdot \overline{B} \cdot \overline{Ci} + A \cdot B \cdot Ci.$
$Co(A, B, Ci) = \overline{A} \cdot B \cdot Ci + A \cdot \overline{B} \cdot Ci + A \cdot B \cdot \overline{Ci} + A \cdot B \cdot Ci.$

c) Nota: para ambas as funções, S e Co, deduzir expressão da função realizada pelo circuito e simplificar. Alternativamente, pode ser construída a tabela de verdade a partir das expressões deduzidas do circuito e das expressões obtidas na alínea anterior, verificando-se que coincidem. Esta segunda opção poderá ser a mais simples.

d) $Co = 1, w_3 = 1, w_2 = 1, w_1 = 0, Ci = 0, S_3 = 1, S_2 = 0, S_1 = 0$ e $S_0 = 0$.

Exercício 20

a)

A_1	A_0	B_1	B_0	$MAIOR$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

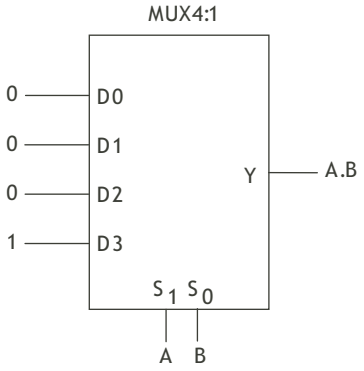
b) $MAIOR = A_1 \cdot \overline{B_1} + A_0 \cdot \overline{B_1} \cdot \overline{B_0} + A_1 \cdot A_0 \cdot \overline{B_0}$.

Exercício 21

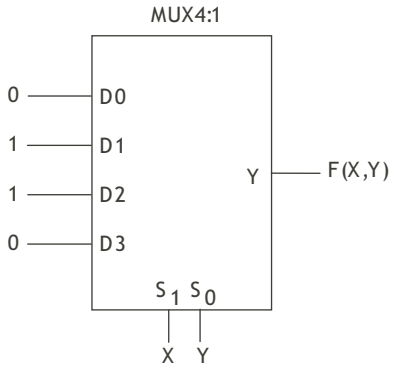
O circuito determina o máximo de A e B.

Exercício 22

a)



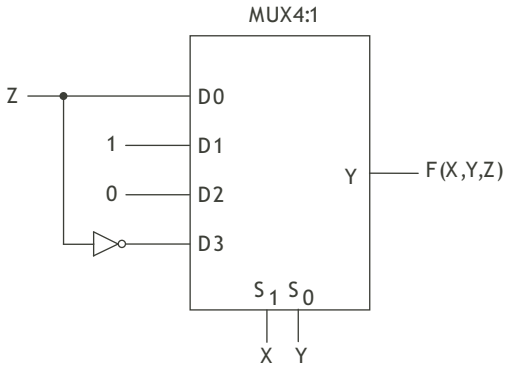
b)



c) Por exemplo, sendo F uma função definida por:

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

então o circuito resultante será:



Exercício 23

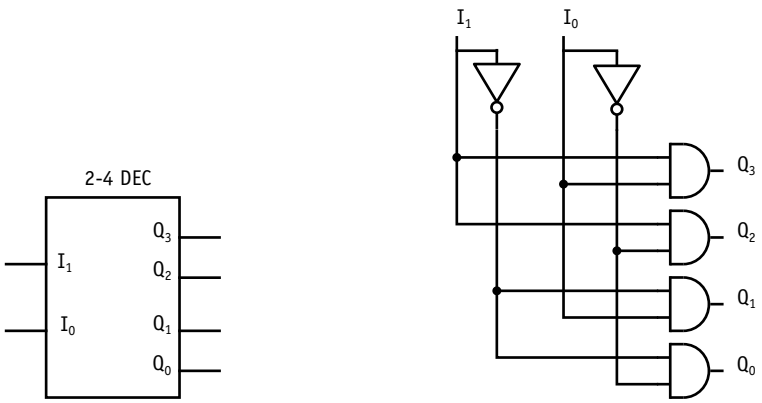
a) $AB = 01 \quad \Rightarrow \quad L_{vm} = 0, L_{lj} = 1 \quad \text{e} \quad L_{vr} = 0.$

$AB = 11 \quad \Rightarrow$ todas as lâmpadas se mantêm desligadas, pois a saída que fica ativa é Y_3 e não é usada.

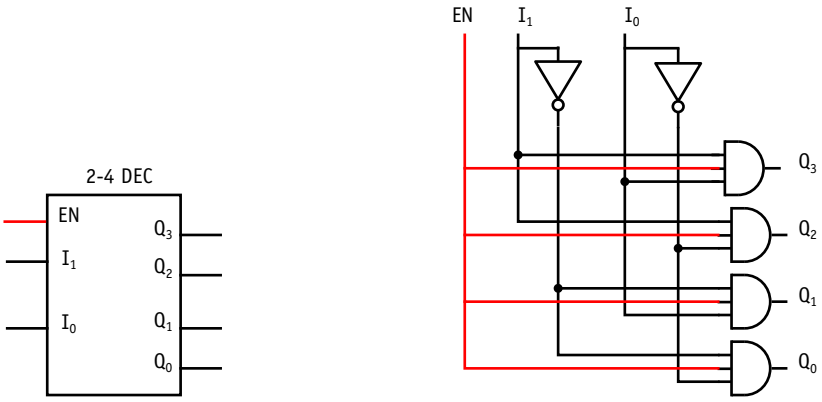
b) É impossível ter simultaneamente duas saídas de um descodificador binário ativas.

Exercício 24

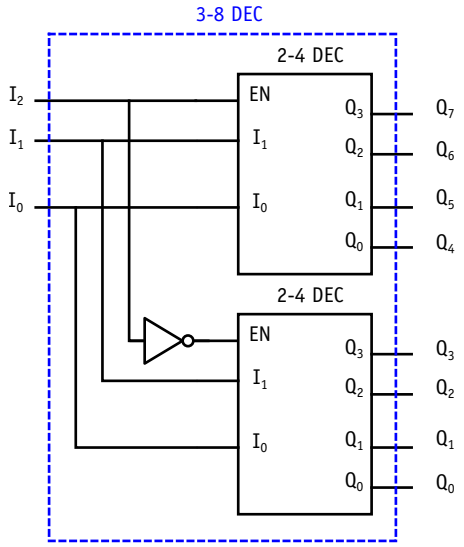
a) O circuito seguinte implementa um descodificador binário 2-para-4.



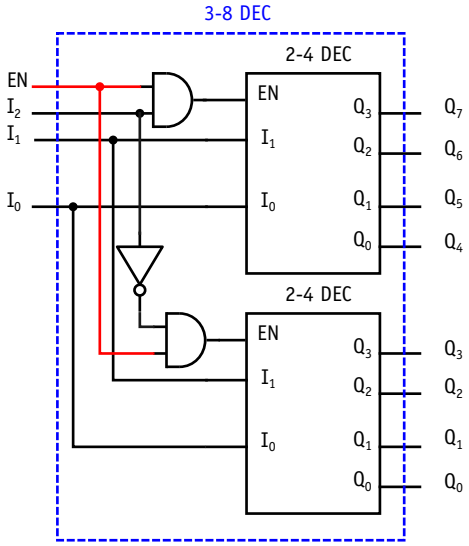
b) O circuito seguinte é um descodificador binário 2-para-4 com sinal de habilitação *(enable)*.



c) O circuito seguinte mostra um descodificador binário 3-para-8 (sem sinal de habilitação). Ter em atenção a numeração das saídas, que depende da forma como a entrada I_2 é ligada.



d) O circuito seguinte mostra um descodificador binário 3-para-8 com sinal de habilitação.



Exercício 25

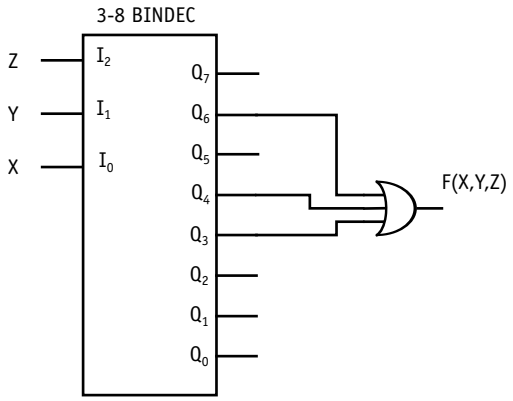
a) $Q_0 = \overline{I_2} \cdot \overline{I_1} \cdot \overline{I_0}$ e $Q_3 = \overline{I_2} \cdot I_1 \cdot I_0$.

b) Ligar X à entrada I_0, Y à entrada I_1 e Z à entrada I_2.

Para $X = 1, Y = 1, Z = 0$, apenas a saída $Q_3 = 1$.

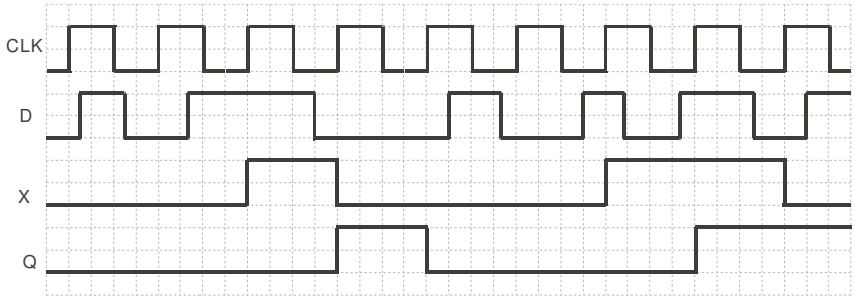
Com $X = 0, Z = 1$ podemos ter uma de duas saídas a 1 (dependendo do valor de Y): Q_4 e Q_6. F deve tomar o valor 1 apenas nestes casos. Logo, $F(X, Y, Z) = Q_3 + Q_4 + Q_6$.

O circuito resultante é:



4 Circuitos sequenciais

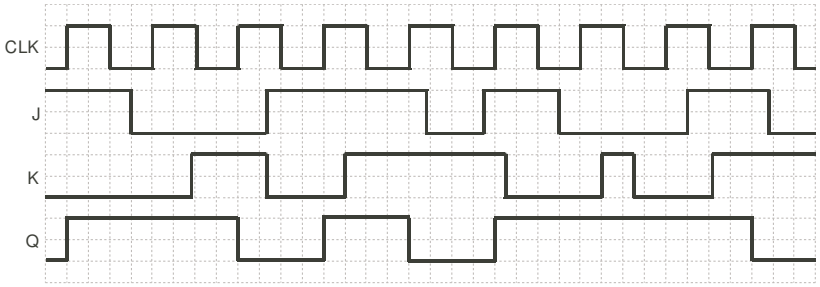
Exercício 6



Exercício 7

a) $D(J, K, Q) = J \cdot \overline{Q} + \overline{K} \cdot Q$

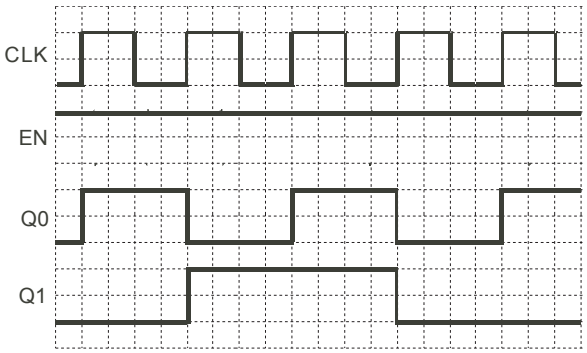
b)



Exercício 8

a) $D_0 = EN \oplus Q_0$ e $D_1 = EN \cdot Q_0 \oplus Q_1$

b) A figura seguinte mostra as formas de onda pedidas.



c) Estado `00` (estado inicial).

Exercício 9

a) 0101001101000010.

b) A saída T assinala as mudanças de valor da entrada X.

c) A saída do contador fornece o número de transições de valor lógico da entrada X, até ao máximo de 15.

Exercício 10

a) 4 bytes.

b) Descodificador binário.

c) O conteúdo do registo 1 passa a ser 1111.

d) A entrada de *enable* permite habilitar a escrita de um valor num registo. A entrada *reset* permite limpar, isto é, escrever o valor 0 num dos registos.

e) O descodificador permite identificar o registo a aceder para uma operação de escrita ou de leitura. Quanto ao multiplexador, permite selecionar a saída de um registo, ou seja, fazer a leitura desse registo.

f) Um segundo multiplexador de 8 (entradas de 4 bits) para 1, assim como uma segunda entrada de 3 bits com o endereço do registo a ler.

Exercício 11

a) [0x4000; 0x5FFF]

Descodificação parcial, pois há um bit (A_{12}) que não é usado pela RAM, originando dois endereços para cada entrada da RAM.

b) 0xCFFF

c) O circuito resulta da expressão $CS_{\text{ROM}} = A_{15} \cdot A_{14} \overline{A_{13}} \cdot \overline{A_{12}} \cdot A_{11}$ (AND com duas entradas negadas).

Exercício 12

a) ROM1: 16 KiB

RAM1: 16 KiB

RAM2: 4 KiB

b) $CS_{\text{ROM1}} = \overline{A_{15}} \cdot \overline{A_{14}}$

$CS_{\text{RAM1}} = \overline{A_{15}} \cdot A_{14}$

$CS_{\text{RAM2}} = A_{15} \cdot A_{14} \cdot \overline{A_{13}} \cdot A_{12}$

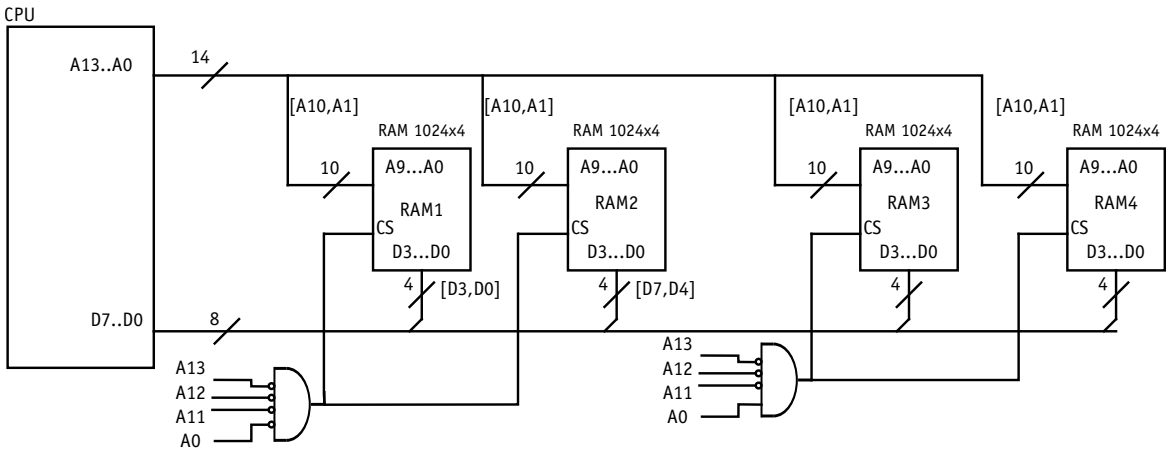
c) —

Exercício 13

Como cada circuito só armazena 4 bits por posição, é necessário usar 2 circuitos para a mesma posição (um para guardar os bits `D[3,0]` e outro para os bits `D[7,4]`. Um grupo de dois circuitos RAM armazena os valores dos endereços pares entre 0 e 2047 (0, 2, 4, ..., 2046), enquanto o segundo grupo guarda os valores dos endereços ímpares (1, 3, 5, ..., 2047).

O bit `A0` deve ser usado para selecionar o grupo de RAMs pretendido.

a) A figura apresenta uma solução. Os circuitos `RAM1` e `RAM2` realizam o grupo que armazena o conteúdo correspondente aos endereços pares.



b) O endereço 250 é par. Logo, os dados vão ficar guardados na posição 250/2=125 dos circuitos `RAM1` e `RAM2`. O valor `A` (bits `[D0,D3]`) é armazenado em `RAM1` e o valor `9` em `RAM2`.

5 Linguagem *assembly*

Exercício 6

É necessário definir uma atribuição arbitrária de variáveis a registos.

a) Atribuição: `f` → `x0`, `g` → `x1`, `j` → `x2`.

```
add       x0, x2, 2
add       x0, x1, x0
```

b) Atribuição: `a` → `x0`, `b` → `x1`, `d` → `x2`, `f` → `x3`, `k` → `x4`.

```
add       x4, x0, x1
sub       x4, x4, x3
add       x4, x4, x2
add       x4, x4, -30
```

c) Atribuição: `f` → `x0`, `g` → `x1`, `h` → `x2`, `A` → `x7`.

```
ldur      x0, [x7, 32]
add       x0, x0, x1
add       x0, x0, x2
```

d) Atribuição: `f` → `x0`, `g` → `x2`, `A` → `x6`, `B` → `x7`.

```
ldur      x5, [x7, 80]
lsl       x5, x5, 3
add       x5, x5, x6
ldur      x0, [x5]
sub       x0, x2, x0
```

e) Atribuição: `f` → `x0`, `g` → `x1`, `h` → `x2`, `k` → `x3`, `A` → `x6`.

```
add       x10, x2, 9
lsl       x10, x10, 3
add       x10, x6, x10
ldur      x0, [x10]
add       x0, x0, x3
sub       x0, x0, x1
```

f) Atribuição: `f` → `x0`, `g` → `x1`, `A` → `x6`, `B` → `x7`.

```
ldur      x10, [x7, 16]
add       x10, x10, 4
lsl       x10, x10, 3
add       x10, x6, x10
ldur      x10, [x10]
sub       x0, x1, x10
```

Exercício 7

a) Atribuição: `f` → `x0`, `g` → `x1`, `h` → `x2`, `i` → `x3`, `j` → `x4`

A expressão correspondente é: `f = f + g + h + i + j`

b) Atribuição: `f` → `x0`, `A` → `x6`

A expressão correspondente é: `f = A[1]`

c) Atribuição: `f` → `x0`, `g` → `x1`, `A` → `x6`

A expressão correspondente é: `f = A[g-3]`

Exercício 8

a) `0x55775577DDFFDDFF`

b) `0x5555555555555550`

c) `0x00000000000000AA`

Exercício 9

a)
```
            mov     X2, X1
            cmp     X1, XZR
            b.ge    pos
            sub     X2, XZR, X2
    pos:  ...
```

Ou, tirando proveito da instrução CSNEG:

```
            cmp     X1, 0
            csneg   X2, X1, X1, ge
```

b)
```
            mov     X1, 0
            cmp     X2, X3
            b.le    pos
            mov     X1, 1
    pos:  ...
```

Ou, tirando proveito da instrução CSINC:

```
            cmp     X2, X3
            csinc   X1, XZR, XZR, le
```

Exercício 10

A cada iteração do ciclo `L1` será guardado em memória (no endereço dado pelo conteúdo do registo `X5`) o valor do registo `X4`. Como em cada ciclo o valor do registo `X4` sofre um deslocamento de 4 bits para a esquerda, o seu valor será zero após 16 iterações e consequentemente a instrução de salto não é realizada, terminando o programa.

Tendo em conta os valores iniciais, a tabela com o conteúdo da memória é:

Endereço	Valor
0x7D0	0x0000000012345678
0x7D8	0x0000000123456780
0x7E0	0x0000001234567800
0x7E8	0x0000012345678000
0x7F0	0x0000123456780000
0x7F8	0x0001234567800000
0x800	0x0012345678000000
0x808	0x0123456780000000
0x810	0x1234567800000000
0x818	0x2345678000000000
0x820	0x3456780000000000
0x828	0x4567800000000000
0x830	0x5678000000000000
0x838	0x6780000000000000
0x840	0x7800000000000000
0x848	0x8000000000000000

Exercício 11

Uma possível solução:

```
1          mov       x0, 0
2  ciclo:  ldur      x1, [x4]
3          cmp       x1, 0
4          b.eq      fim
5          add       x4, x4, 8
6          add       x0, x0, 1
7          b         ciclo
8  fim:    ...
```

Exercício 12

a) Fragmento 1: 20; fragmento 2: 200.

b) Fragmento 1: $5 \times N + 2$; fragmento 2: $(1 + 3 \times 10 + 2) \times N = 33 \times N$.

Exercício 13

Uma possível solução:

```
1              mov   X6, 100      // dimensão das seq.
2              mov   X0, 1
3  proximo:    ldur  X1,[X4]      // extrai elemento de A
4              ldur  X2,[X5]      // extrai elemento de B
5              cmp   X1,X2        // compara os elementos extraidos
6              b.eq  continua     // continua se forem iguais
7              mov   X0,0         // termina se forem diferentes
8              b     fim
9  continua:   subs  X6,X6, 1
10             b.eq  fim
11             add   X4,X4, 8     // se não chegou ao fim
12             add   X5,X5, 8     // passar ao próximo elemento
13             b     proximo
14 fim:        ...
```

Exercício 14

Uma solução entre outras possíveis:

```
1          mov    X8, 50          // contador: X8
2          mov    X7, 0           // resultado a zero
3  ciclo:  ldur   X9, [X0]
4          ands   X10, X9, 1
5          b.eq   prox            // par
6          add    X7, X7, 1       // ímpar
7  prox:   add    X0, X0, 8       // próximo endereço
8          subs   X8, X8, 1       // decrementar contador
9          b.ne   ciclo
10         ...
```

Exercício 15

a) Parâmetros da sub-rotina:

- `X0`: endereço-base da sequência
- `X1`: número de elementos da sequência

```
1  soma:   mov   X2, 0        // coloca contador a 0
2  L1:     cmp   X1, 0        // verifica se chegou ao fim
3          b.eq  L2           // terminar
4          ldur  X3, [X0]     // obter um elemento
5          add   X2, X2, X3   // acumular
6          add   X0, X0, 8    // endereço do próximo elemento
7          add   X1, X1, -1   // ajustar nº de elementos
8          b     L1           // repetir (próximo elemento)
9  L2:     mov   X0, X2       // devolver resultado em X0
10         ret
```

b) Parâmetros da sub-rotina:

- `X0`: endereço-base da sequência

- `X1`: número de elementos da sequência

```
1  impar:  mov    X7, 0           // coloca contador a 0
2  L21:    cmp    X1, 0           // verifica se chegou ao fim
3          b.eq   L23             // terminar
4          ldur   X9, [X0]        // obter um elemento
5          ands   X10, X9, 1      // testar se é impar
6          beq    L22             // se par, não contabiliza
7          add    X7, X7, 1       // contabilizar
8  L22:    add    X0, X0, 8       // endereço do próximo elemento
9          sub    X1, X1, 1       // ajustar nº de elementos
10         b      L21             // repetir (próximo elemento)
11 L23:    mov    X0, X7          // devolver resultado em X0
12         ret
```

c) Parâmetros da sub-rotina:

- `X0`: endereço-base da sequência
- `X1`: número de elementos da sequência
- `X2`: valor de limiar

```
1  limiar: mov    X7, 0           // coloca contador a 0
2  L31:    cmp    X1, 0           // verifica se chegou ao fim
3          b.eq   L33             // terminar
4          ldur   X9, [X0]        // obter um elemento
5          cmp    X9, X2          // testar se é maior que o limiar
6          b.lt   L32             // se menor, não contabiliza
7          add    X7, X7, 1       // contabilizar
8  L32:    add    X0, X0, 8       // endereço do próximo elemento
9          sub    X1, X1, 1       // ajustar nº de elementos
10         b      L31             // repetir (próximo elemento)
11 L33:    mov    X0, X7          // devolver resultado em X0
12         ret
```

d) Parâmetros da sub-rotina:

- `X0`: endereço-base da sequência 1
- `X1`: endereço-base da sequência 2
- `X2`: número de elementos de cada sequência

```
1  iguais: mov   X7, 1          // coloca resultado a 1
2  L41:    cmp   X2, 0          // verifica se chegou ao fim
3          b.eq  L43            // terminar
4          ldur  X9, [X0]       // obter um elemento de cada sequência
5          ldur  X10,[X1]
6          cmp   X9, X10
7          b.ne  L42            // Se nao forem iguais termina
8          add   X0, X0, 8      // endereço do próximo elemento
9          add   X1, X1, 8
10         sub   X2, X2, 1      // ajustar nº de elementos
11         b     L41            // repetir (próximo elemento)
12 L42:    mov   X7, 0          // resultado a 0
13 L43:    mov   X0, X7         // devolver resultado em X0 (1 se iguais)
14         ret
```

6 Organização de um CPU

Exercício 7

Ter em atenção que `PcSrc = AND(Zero, Branch)`.

a) `RegWrite=1, MemRead=1, ALUSrc=1, MemWrite=0, MemtoReg=1, PCSrc=0, ALUOp=00, Reg2Loc=X.`

Todos os componentes realizam trabalho útil, exceto `Shift left 2` e o somador para endereço de saltos (`Add`). O valor `Read data 2` também não é utilizado.

b) `RegWrite=0, MemRead=0, ALUSrc=1, MemWrite=1, MemtoReg=X, PCSrc=0, ALUOp=00, Reg2Loc=1.`

Todos os componentes realizam trabalho útil, exceto `Shift left 2`, somador para endereços de saltos (`Add`) e multiplexador `MemtoReg`. utilizado.

c) `RegWrite=0, MemRead=0, ALUSrc=0, MemWrite=0, MemtoReg=X, ALUOp=01, Reg2Loc=1.`

$$\texttt{PCSrc} = \begin{cases} 0 & \text{se } \texttt{X9} \neq 0 \\ 1 & \text{se } \texttt{X9} = 0 \end{cases}$$

Todos os componentes realizam trabalho útil, exceto memória de dados (`D-Mem`) e multiplexador `MemtoReg`. O valor `Read data 1` também não é utilizado.

Exercício 8

a) Instrução `CBZ`.

b) Instrução `STUR`.

Exercício 9

Acrescentar um multiplexador que receberá na entrada 0 a saída proveniente do multiplexador controlado por `PCSrc` e na entrada 1 a saída `Read data 1` do banco de registos. Este novo multiplexador deverá ser controlado por um sinal adicional de controlo, `BranchReg`, tal que `BranchReg=1` se opcode=`11010110000` e `BranchReg=0` no caso contrário.

Exercício 10

a) `0xF8428228`.

b) 17. Sim.

c) Indefinido (`Reg2Loc=X`). Não.

d) 8. Sim.

e) `MemRead=1, MemWrite=0`.

f) `0xB400000B`; 0, não; 11, sim; 11, não; `MemRead=0, MemWrite=0`.

Exercício 11

a) `I-Mem` → `Regs` → `ALU` → `D-Mem` → `Mux` . Tempo: 1100 ps.

b) `I-Mem` → `Control` → `Mux` → `Regs` → `Mux` → `ALU` → `Mux (PCSrc)`. Tempo: 910 ps.

Exercício 12

a) 320 ps.

b) `RegWrite`. 700 ps para a instrução mais lenta `(LDUR)`.

c) `Reg2Loc`. 120 ps. Evita que `STUR` demore mais que 1100 ps, correspondente à instrução mais lenta (`LDUR`).

7 Memória Cache

Exercício 3

a) Etiqueta: 12 bit; índice: 4 bit; espaço de endereçamento: `0x00000 - 0x3ffff`; capacidade máxima: 256 KiB.

b)
1. Lê valor `0x6198fa34` de memória *cache*.
2. Lê valor de memória principal e coloca-o na posição 5, com v=1 e etiqueta `0x8d1`.
3. Lê valor de memória principal e coloca-o na posição 12, com v=1 e etiqueta `0xfde`.
4. Escreve valor `0x1212abab` na posição 1, v=1, e atualiza a memória principal no endereço `0x048c4`.
5. Não altera memória *cache* (não existe informação em *cache* para este endereço, porque a etiqueta é diferente); coloca valor `0x00001111` em memória principal no endereço `0x3f7d0`.
6. Não altera memória *cache* (não existe informação válida em *cache* para este endereço); coloca valor `0xaaaabbbb` em memória principal no endereço `0x1dde8`.

Exercício 4

a) O campo `d` indica se o valor de conteúdo em memória *cache* é diferente do valor em memória principal.

b) A determinação do índice e da etiqueta faz-se como no problema anterior.

1. Índice 4, etiqueta `0xb7c` e `v=1`: *read hit*. Ler valor `0x6198fa34` de memória *cache* (bloco 4).
2. Endereço igual ao da alínea anterior: *write hit*. Como `d=0`, não é necessário fazer *write back*. O valor `0x33334444` é escrito na memória *cache* (bloco 4), `v=1`, `d=1`, etiqueta `0xb7c`.
3. Índice 9, etiqueta `0x431`, `v=1`: *write miss*. Como `v=1` e `d=1`, é necessário fazer *write back*: escrever o valor `0xffffeffe` (posição 9) em memória (endereço `0x21de4`). O valor `0x9999aaaa` é colocado na posição 9 da memória *cache*, com `v=1`, `d=1`, etiqueta `0x431`.
4. Índice 5, etiqueta `0x8d1` e `v=0`: *read miss*. Como `v=0`, não faz *write back* (valor de `d` não interessa neste caso). Ler valor de memória principal e colocá-lo na posição 5, com `v=1`, `d=0`, etiqueta `0x8d1`.
5. Índice 12, etiqueta `0xfde`, `v=0`: *read miss*. Como `v=0`, não faz *write back*. Ler valor de memória principal e colocá-lo na posição 12, com `v=1`, `d=0`, etiqueta `0xfde`.
6. Índice 9, etiqueta `0x877`. O bloco 9 foi alterado pelo terceiro acesso, pelo que a etiqueta é diferente (etiqueta atual do bloco `0x431`). Como `v=1`, ocorre um *write miss*. Como `d=1`, é necessário efetuar *write back*: escrever o valor valor `0x9999aaaa` na posição de endereço `0x00010c64`. Escrever valor `0xbbbb7777` no bloco 9 da memória *cache*, `v=1`, `d=1`, etiqueta `0x877`.
7. Índice 7, etiqueta `0xabf`, `v=1`: como as etiquetas não coincidem, ocorre *write miss*. Como `d=0`, não existe necessidade de fazer *write-back*. Colocar o valor `0x1212abab` em memória *cache* (bloco 7) com `v=1`, `d=1`.
8. Índice 12, etiqueta `0xfde`, `v=1` (mesmo endereço que no quinto acesso, que alterou este bloco): *read hit*. Basta atualizar valor em memória *cache*: escreve valor `0x00001111` no bloco 12 com `v=1`, `d=1`, etiqueta `0xfde` (mantém etiqueta).

Exercício 5

a) 59 bits

b) `0x2a5d04aa`

c) O conteúdo do bloco 0 é alterado para `2a 5d 04 aa 00 00 00 00`.

d) `0x000000cc`

Exercício 6

a) P1: 1,61 GHz; P2: 1,52 GHz.

b) P1: 8,6 ns; P2: 6,26 ns.

c) Ter em atenção que existem $1 + 0{,}36$ acessos a memória por instrução.
P1: CPI$=1 + 0{,}114 \times 1{,}36 \times 70/0{,}62=18{,}5$; P2: CPI$=12{,}54$.

Exercício 7

a) 90 faltas/10^3 instruções.

b) 6,43 %.

c) Um acesso a memória custa 80 ciclos (período de relógio: 1 ns).

i) $\text{CPI}_{\text{efetivo}} = 1{,}2 + 1{,}4 \times 80 = 113{,}2$ ciclos/instrução.

ii) $\text{CPI}_{\text{efetivo}} = 1{,}2 + 0{,}0643 \times 1{,}4 \times 80 = 8{,}4$ ciclos/instrução.

8 Desempenho

Exercício 9

a) $\text{IPC}_{P_1} = 1{,}43$ (instruções/ciclo)
$\text{IPC}_{P_2} = 2{,}00$ (instruções/ciclo)
$\text{IPC}_{P_3} = 3{,}33$ (instruções/ciclo)

b) 2,1 GHz

c) 27×10^9

Exercício 10

a) P_1: $t_{exec} = 1{,}87$ ms $\quad$ P_2: $t_{exec} = 1{,}00$ ms
P_2 é o processador mais rápido.

b) $\text{CPI}_{P_1} = 2{,}8$ (ciclos/instrução)
$\text{CPI}_{P_2} = 2{,}0$ (ciclos/instrução)

c) P_1: $2{,}8 \times 10^6$ $\quad$ P_2: 2×10^6

Exercício 11

P_1: 4×10^9 (instruções/segundo) $\quad$ P_2: 3×10^9 (instruções/segundo)

Exercício 12

a) C é o computador mais rápido (45 s).

b) C é 1,67 vezes mais rápido que B e 4,93 vezes mais rápido que A.

Exercício 13

a) $t_{exec} = 675$ ns

b) CPI = 1,93 (instruções/segundo)

c) A execução é 1,23 mais rápida e CPI = 1,69

Exercício 14

88 %

Exercício 15

a) 1,67

b) 5

Exercício 16

a) Desenhar o gráfico da função

$$S(f) = \frac{1}{\frac{f}{10} + (1 - f)} = \frac{10}{10 - 9 \times f} \qquad 0 \leq f \leq 1$$

b) $f = \frac{5}{9} \approx 0{,}56 = 56\,\%$

c) Percentagem de tempo gasto em modo vetorial após introdução da alteração: $\frac{1}{9} \approx 11\,\%$

d) *Speedup* máximo: 10; para obter *speedup* de 5 é preciso que $f = \frac{8}{9} \approx 88{,}9\,\%$

e) Duplicação de desempenho da unidade vetorial: $S \approx 2{,}99$. Sem duplicação de desempenho, o mesmo *speedup* é obtido para $f \approx 0{,}74 = 74\,\%$. Provavelmente é mais fácil passar de 70 % para 74 % do que duplicar o desempenho da unidade vetorial.

Exercício 17

O ganho de rapidez da nova versão é 2.