

KLS

Sistemas Digitais e Microprocessadores

Sistemas Digitais e Microprocessadores

Victor Perim

Giancarlo Michelino Gaeta Lopes

Andrea Acunha Martin

© 2020 por Editora e Distribuidora Educacional S.A.

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Editora e Distribuidora Educacional S.A.

2020

Editora e Distribuidora Educacional S.A.

Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: <http://www.kroton.com.br/>

Sumário

Unidade 1	
Circuitos digitais e álgebra booleana	7
Seção 1	
Bases numéricas e códigos	9
Seção 2	
Introdução à álgebra booleana.....	25
Seção 3	
Circuitos digitais.....	41
Unidade 2	
Circuitos lógicos combinacional e sequencial	62
Seção 1	
Introdução à lógica combinacional e sequencial.....	64
Seção 2	
Circuitos combinacionais	81
Seção 3	
<i>Flip-flops</i> e circuitos correlatos.....	107
Unidade 3	
Arquitetura de microprocessadores e microcontroladores.....	137
Seção 1	
Introdução ao microprocessamento	139
Seção 2	
Arquitetura RISC.....	160
Seção 3	
Arquitetura CISC.....	171
Unidade 4	
Programação de microprocessadores e microcontroladores.....	184
Seção 1	
Microcontrolador: Arduino	186
Seção 2	
Novas arquiteturas e tendências	206
Seção 3	
Aplicações para processadores ARM	225

Palavras do autor

Caro aluno, você provavelmente já usa e está bem acostumado com os aparelhos eletrônicos que nos rodeiam por todos os lados. Apesar disso, você deve saber que as gerações anteriores viveram realidades completamente diferentes das atuais e tiveram, em algum momento, que se adaptar a esse modo de vida em que as tecnologias estão cada vez mais integradas, em que mais e mais pessoas interagem com elas.

A proposta deste curso é que, de maneira gradual, você passe a visualizar os sistemas como um desenvolvedor.

Imaginamos que você já tenha ouvido falar do termo *Arduino* (ou microcontrolador) e esteja muito curioso para entender como essas novas máquinas são usadas em projetos de automação.

No amplo campo em que a tecnologia se ramifica, quando se trata de computação e eletrônica, existe uma separação básica: o *hardware*, ou seja, a parte física do sistema, e o *software*, a parte não física (ideia lógica ou abstração). Este curso traz o desafio especial de diminuir a distância entre esses dois elos: a parte que vemos (o *hardware*) com a parte que não vemos (o *software*).

Sendo assim, ao fim deste curso, cumprindo todas as tarefas, você estará pronto para desenvolver projetos de eletrônica, com processamento de dados, controle de periféricos externos e comunicação entre dispositivos. Muitas áreas referentes à automação e à eletrônica atuam com microcontroladores, e cada vez mais o profissional que domina bem essas áreas é mais requisitado no mercado de trabalho, principalmente na área de desenvolvimento de projetos de sistemas embarcados, uma vez que estamos entrando na chamada “Quarta Revolução Industrial” (também denominada de *Indústria 4.0*), em que os sistemas ciberfísicos assumem autonomia para tomar decisões e se comunicar tanto entre si quanto com os humanos.

De todos esses avanços, uma coisa é certa: se você tem real vontade de aprender sobre sistemas embarcados, agora não lhe faltará mais nada para isso! Este livro, unido à energia e à determinação de seus estudos, pavimentarão o caminho para você se tornar um excelente profissional, capaz de resolver muitos problemas de automação.

É com esse espírito que este livro foi escrito com muito empenho e dedicação e que todos os detalhes foram pensados para que você, prezado aluno, possa aprender de maneira agradável e eficaz essa disciplina tão instigante.

Como você pode perceber, este material está dividido em quatro unidades.

Na Unidade 1, iniciaremos o estudo com circuitos digitais e álgebra booleana, apresentando os conceitos fundamentais sobre sistemas lógicos, sua história, as principais bases numéricas e conversão de bases. Em seguida serão abordados alguns códigos digitais, como o BCD, ASCII e código de Gray, bem como algumas aplicações primárias. Após estudarmos as portas lógicas e suas operações, veremos como projetar e implementar circuitos digitais para circuitos sem memória mediante pequenos exemplos práticos.

Na Unidade 2, vamos abordar as características e diferenças do circuito lógico combinacional e sequencial. Também veremos o que são sistemas analógicos e sistemas digitais. Para os digitais, que é o foco deste curso, estudaremos a classificação da lógica de circuitos entre combinacional, que é estático, e sequencial, que conta com um fluxograma para ser projetado. Em seguida estudaremos os circuitos aritméticos, comparadores, multiplexadores e as respectivas aplicações, finalizando a unidade com a construção de máquina de estados.

Por sua vez, na Unidade 3, aprofundaremos nosso estudo em arquitetura de microprocessadores e microcontroladores, com os dois principais padrões de arquitetura: Von Newman; e Havard. Também veremos os principais elementos internos de um computador e a maneira como eles operam na execução de uma aplicação. Estudaremos na sequência os tipos de arquitetura RISC e CISC bem como suas particularidades.

Na Unidade 4, abordaremos a programação de microprocessadores e microcontroladores. Nela vamos tratar de sistemas computacionais em termos mais práticos, com aplicações em problemas reais. Veremos também como criar programas em linguagem C para o Arduino e como montar projetos práticos por meio de periféricos externos.

Ao final do curso esperamos que você, aluno, esteja pronto para iniciar a carreira no mercado de trabalho, principalmente se for da área técnica, sobretudo em relação ao desenvolvimento de projetos embarcados. Para atingirmos esse objetivo, faltam agora apenas empenho, disciplina e dedicação. No entanto, não pense se tratar de “coisa de outro mundo” ou um “bicho de sete cabeças”. Estamos certos de que será uma ótima experiência, até mesmo prazerosa, pois este material foi escrito com a sua trajetória de estudos em mente.

Vamos começar?

Unidade 1

Victor Perim

Circuitos digitais e álgebra booleana

Convite ao estudo

Estimado aluno, nesta primeira unidade teremos os primeiros contatos com o sistema binário e com a lógica digital, que é a base da construção dos sistemas digitais e computacionais. Vamos aprender na Seção 1.1, como os dados são convertidos em sinais e como esses sinais podem ser manipulados para gerar um resultado útil. É muito importante um bom aproveitamento no estudo dessa primeira parte, pois os fundamentos e todos os outros assuntos posteriores remeterão, direta ou indiretamente, aos termos apresentados aqui. Depois de conhecer também um pouco sobre a história e a evolução do computador, vamos ver como usamos as bases numéricas para representar as grandezas, bem como as conversões entre elas. Na sequência vamos ver alguns códigos importantes, como o código BCD, usado para tratar os números decimais de forma binária, e, por exemplo, o ASCII, usado para tratar texto.

Em seguida, na Seção 1.2, vamos entender como funcionam as operações com álgebra booleana, que é a base da lógica digital, depois vamos passar pelas portas lógicas, pelas formas de simplificação das suas expressões e vamos finalizar com os mapas de Karnaugh.

Com a intenção de aproximar os conceitos teóricos da Seção 1.3 às práticas nas indústrias, exploraremos o seguinte cenário: imagine uma grande granja onde está sendo criando um novo setor inteiramente automatizado para o qual há uma vaga de engenheiro desenvolvedor, que deverá ser responsável pelo sistema de controle de fluxo das galinhas no bloco de banho de luz e alimentação.

Os circuitos lógicos serão projetados com base em tabelas-verdade devidamente construídas para essa aplicação, pela simplificação algébrica e pelo modo como seu circuito deverá ser gerado. Por fim, vamos ver como implementar circuitos lógicos com base em expressões booleanas e também como obter a expressão booleana por um circuito lógico.

Esses conceitos abordados compõem todo o conteúdo teórico necessário para você adquirir as competências previstas para esta unidade, além de capacitá-lo a resolver as situações-problemas enfrentadas.

Ao final desta unidade você será capaz de avaliar os requisitos mínimos do sistema, bem como as partes que compõem o circuito de controle automático, capaz de operar a granja adequadamente.

Considere agora que você é o engenheiro contratado para ser o responsável pela parte de eletrônica digital desse novo projeto, devendo estar preparado para tomar as decisões mais importantes.

Estamos certos de que este será um ótimo estudo, esclarecedor e estimulante, porém precisamos, antes, do embasamento teórico destinado a entender corretamente como a prática será construída.

Preparado? Então vamos lá.

Seção 1

Bases numéricas e códigos

Diálogo aberto

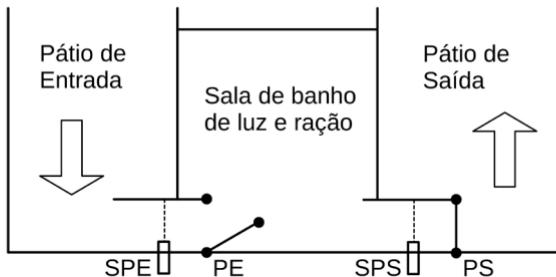
Estimado aluno, como vivenciamos uma era de total conectividade, você provavelmente deve estar familiarizado com aparelhos eletrônicos do ponto de vista de usuário. O propósito aqui é que você comece a conhecer os sistemas como um engenheiro desenvolvedor.

Inicialmente, aproveitando alguns conhecimentos já adquiridos no curso, estudaremos e entenderemos como é o funcionamento básico de um sistema digital, bem como sua composição.

Para consolidar esse primeiro aprendizado, consideraremos a situação proposta na apresentação da unidade, com base na qual você colocará em prática os conceitos estudados.

Nesse cenário, voltamos ao exemplo da granja, onde, como já destacamos, está sendo criado um novo setor inteiro automatizado para o qual há uma vaga de engenheiro desenvolvedor, que será responsável pelo sistema de fluxo de galinhas no bloco de banho de luz e alimentação. Esse processo é demonstrado na Figura 1.1. Nela, veja que inicialmente a porta de entrada (PE) está aberta; e a porta de saída (PS), fechada. As galinhas que vêm do pátio de entrada entram, uma a uma, na sala de banho e alimentação. Essas galinhas são contabilizadas pelo sensor de presença na entrada (SPE), que gera um pulso digital para cada contagem.

Figura 1.1 | Esquema da planta do novo setor da granja

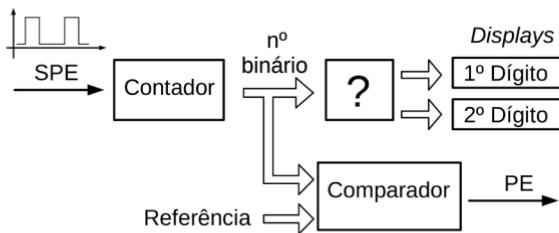


Fonte: elaborada pelo autor

Nessa primeira parte do projeto, será feito o arranjo geral dos blocos lógicos empregados no sistema. O sensor digital de presença será incrementado por um contador, e o valor correspondente ao número de galinhas será

codificado em BCD, para ser impresso em dois *displays* de 7 segmentos, a fim de indicar o valor aos operadores.

Figura 1.2 | Diagrama de blocos lógicos do circuito digital



Fonte: elaborada pelo autor.

Agora reflita sobre o seguinte cenário: sabendo que o número de galinhas que devem entrar em cada sessão de banho de luz e ração é 30 e que um comparador é utilizado para contar as galinhas e fechar a porta de entrada para começar o processo, responda:

- Sabendo que a referência é 30, qual é o valor binário que aparece na contagem quando a porta de entrada (PE) é fechada?
- Qual deve ser o bloco lógico (indicado com uma interrogação na figura) para que os *displays* possam operar adequadamente?

Você já tem alguma ideia?

Não pode faltar

História da eletrônica digital

Desde os primórdios, o ser humano tem, na medida do possível, adaptado a natureza a seu redor para atender às próprias necessidades. Segundo Pereira e Rocha Filho (2017, p. 11), matemáticos e estudiosos já conceberam a ideia do digital muito antes de perceberem a aplicabilidade que teria em um futuro distante. Assim, parece-nos um tanto óbvio que o avanço da tecnologia foi o principal responsável por essa escalada, cujo foco central foram os sistemas de automação. No entanto, toda essa tecnologia é fruto de muito estudo e pesquisa, tendo tido início muito anos antes de qualquer dispositivo eletrônico ter sido confeccionado.

O termo *digital* tem origem na forma com que os computadores operam – mediante dígitos (Floyd, 2007, p. 19). Os sistemas que inicialmente eram

apenas mecânicos ou hidráulicos passaram a ser eletromecânicos (por relés), depois se tornaram eletrônicos e, atualmente, são todos computadorizados, ou seja, operam por *software*. “Apesar de não usarmos em projetos atuais circuitos digitais “puros” (ou em forma bruta), estes são a base para a construção de sistemas mais completos, como microprocessadores, micro-controladores e os computadores.” (NASCIMENTO, 2017, p. 12)

Reflita

Não é possível obter uma regra geral para classificar as grandezas, o que é feito por nossa inspeção e experiência de vida. Por exemplo, grandezas como a temperatura de um ambiente ou a altura de uma pessoa, como sabemos, são valores contínuos, apesar de usarmos números discretos para representá-las, pois não usamos infinitas casas após a vírgula. A natureza da grandeza não restringe a quantidade de casas após a vírgula. O homem faz isso para poder se comunicar. Por sua vez, outras grandezas, como a quantidade de filhos de um casal, a idade de uma pessoa, etc., são naturalmente discretas.

Grandezas analógicas e digitais

Os circuitos elétricos e eletrônicos, assim como as grandezas da natureza, podem ser divididos em duas grandes categorias: os *discretos* ou os *contínuos*. A eletrônica digital envolve grandezas com valores discretos, e a eletrônica analógica envolve grandezas com valores contínuos.

A maioria do que se pode medir quantitativamente na natureza se encontra na forma analógica. No entanto, alguns sinais ou valores são inquestionavelmente discretos, como o número de filhos de um casal, a quantidade de quartos em uma casa ou o resultado do lançamento de um dado em jogos de azar.

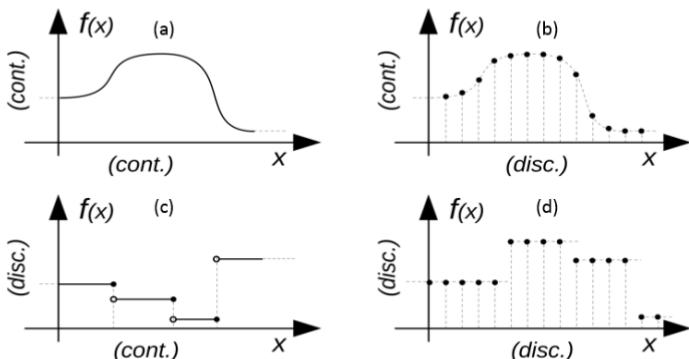
Assimile

O termo *análogo* é equivalente a **contínuo**, e o termo *digital* é equivalente a **discreto** (Tocci, 2011, p.4).

Devemos atentar para as possíveis combinações: um sinal pode ser discreto ou não em amplitude, e discreto ou não no tempo (ou espaço) de forma independente. A Figura 1.3 mostra essas quatro combinações. Na parte superior esquerda da imagem, um sinal, que representamos (ou modelamos) mediante uma função, pode ser contínuo no tempo (domínio da função). O domínio desse sinal se torna discreto pelo processo chamado

de **amostragem**. Esse processo é basicamente a amostragem, ou leitura, do sinal analógico nos instantes de amostragem, que vai gerar uma coleção finita de valores, como um vetor. Além disso, o sinal pode ser contínuo em magnitude, que são os dois gráficos (a) e (b) da Figura 1.3, sendo convertido para discreto através do processo de **quantização**, visto nos gráficos (c) e (d) da mesma ilustração. Esse processo consiste em encontrar quais dos pontos discretos está mais próximo de cada valor de magnitude, que é substituído pelo ponto mais próximo. Assim, uma quantidade finita de valores é usada para a magnitude do sinal, que agora é chamado de *quantizado*.

Figura 1.3 | Tipos de sinais: combinações entre contínuos e discretos



Fonte: elaborada pelo autor.

“É comum encontrar sistemas que utilizem, em certas partes, sinais contínuos e, em outras partes, sinais discretos, como o sinal de um microfone, que é ‘discretizado’ em tempo e amplitude para ser armazenado, processado ou transmitido digitalmente.” (PEDRONI, 2010, p. 25)

Dígitos binários, níveis lógicos e formas de ondas digitais

O termo *digital* para a eletrônica se refere a circuitos e sistemas nos quais existem apenas dois estados possíveis, considerando sinais individuais. Para essa representação, são usados dois níveis de tensão diferentes: um ALTO e um BAIXO, não sendo admitidos valores intermediários. Os dois estados também podem ser representados por níveis de corrente, bits e ressaltos num CD ou DVD, etc.

“Em sistemas digitais, tais como computadores, as combinações de dois estados, denominadas códigos, são usadas para representar números, símbolos, caracteres alfabéticos e outros tipos de informações. O sistema de

numeração de dois estados é denominado de binário, e os seus dois dígitos são 0 e 1. Um dígito binário é denominado de bit.” (TEIXEIRA; TAVARES; PEREIRA, 2017, p. 13)

Diferença entre dado e sinal

Essa é uma confusão muito comum entre os estudantes, uma vez que a palavra *informação* está presente em ambos (*dado* e *sinal*), embora, na verdade, sejam dois conceitos distintos que estão intrinsecamente interligados (FOROUZAN, 2008, p. 57).

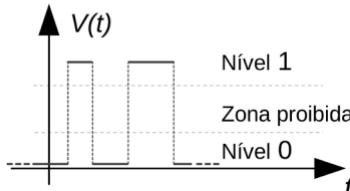
Assimile

Os dados são a informação de forma pura, sem considerar a maneira como esta está sendo transmitida. Os sinais são as grandezas da natureza (elétricas, mecânicas, térmicas) que estão representando algum dado. Por exemplo, se alguém fala (para outra pessoa) que tem 26 anos de idade, esse valor se refere ao dado; por sua vez, o sinal seria a onda sonora que atingiu o ouvido do receptor, que recupera o dado. Ou para você que está lendo, o sinal, nesse caso, seria a tinta no papel, ou os pixels de uma tela.

Níveis lógicos

As tensões usadas para representar 1 e 0 são denominados níveis lógicos. Teoricamente, um nível de tensão representa um nível ALTO e o outro representa um nível BAIXO (PEREIRA; TEIXEIRA, 2018, p. 16). Entretanto, em um circuito digital prático, um nível ALTO pode ser qualquer tensão entre um valor mínimo e um valor máximo estipulado, como podemos observar na Figura 1.4:

Figura 1.4 | Tensão e sinais lógicos



Fonte: elaborada pelo autor.

A história nos ensina que, inicialmente, foi utilizado o sistema decimal para desenvolver os primeiros computadores. No entanto, ocorriam tantos erros de transmissão por conta do ruído e da grande proximidade entre os

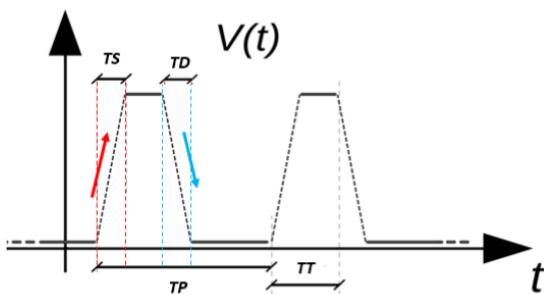
dez níveis de tensão, que esses níveis foram sendo reduzidos até chegarem ao seu menor valor: dois. Portanto, mesmo que se transmitisse menos informação de cada vez, a chance de sucesso era tão mais alta que compensava a utilização do método binário.

Além disso, todo sinal digital (assim como os analógicos) está sujeito a fenômenos de degradação, que podem causar efeitos indesejados pela distorção do sinal original.

Exemplo do sistema de telégrafo

As principais características do sinal digital, que é transmitido mediante pulsos binários, podem ser observadas na Figura 1.5: tempo de subida (T_s); tempo de descida (T_d) (geralmente igual ao de subida); largura do pulso ou período/tempo de trabalho (T_T) (*duty time*), que pode ser dado em termos de valor absoluto ou porcentagem. Este é periódico para sinais de relógio (*clock*), e não periódico para sinais de dados. Por fim, há o período (T_p), que define a frequência do sinal. “Para sinais digitais não periódicos fica difícil determinar o período, mas geralmente o tamanho do bit é fixo, definindo a frequência (máxima) do sinal.” (TOCCI; WIDMER; MOSS, 2011, p. 3)

Figura 1.5 | Características de um pulso digital



Fonte: elaborada pelo autor.

Operações com bases numéricas

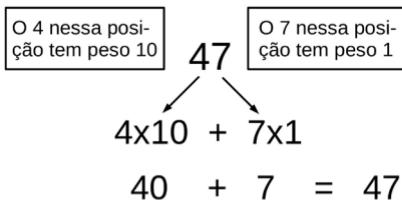
Há muitos sistemas de numeração usados tanto na matemática pura como na tecnologia digital. O sistema de numeração binário e os códigos digitais são fundamentais para os computadores e para a eletrônica digital em geral.

Vamos estudar agora o sistema de numeração binário e suas relações com outros sistemas de numeração, como o decimal, o hexadecimal e o octal.

Números decimais

No sistema de numeração decimal, cada um dos dígitos, de 0 a 9, representa uma certa quantidade. Como sabemos, os dez símbolos (dígitos) não nos limita a expressar apenas dez quantidades diferentes porque podemos usar vários dígitos posicionados adequadamente formando um número para indicar a magnitude (módulo) da quantidade. Podemos expressar quantidades até o número nove, antes de usar mais dígitos; se queremos expressar uma quantidade maior que nove, usamos dois ou mais dígitos, e a posição de cada dígito dentro do número nos diz a magnitude que ele representa. Se, por exemplo, queremos expressar a quantidade *quarenta e sete*, usamos (pelas respectivas posições no número) o dígito 4 para representar a quantidade *quarenta* (a dezena) e o dígito 7 para representar a quantidade *sete* (a unidade), conforme pode ser visto na Figura 1.6.

Figura 1.6 | Representação numérica na base decimal



Fonte: Floyd (2007, p. 64).

Outro exemplo: o número 2451 pode ser representado pela expressão:
 $2 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 1 \times 10^0 =$

$$= 2 \times 1000 + 4 \times 100 + 5 \times 10 + 1 \times 1 = 2451_{10}$$

O fato de usarmos o valor 10 com suas potências inteiras determina a base da representação do número, cujos algarismos entram multiplicando cada uma das potências de 10, específica para cada posição deles.

Essa forma nos serve quando precisamos lidar com números inteiros; no entanto, quando se trata de números fracionários, aparecem também potências de 10 com expoente negativo, que diminuem da esquerda para a direita começando com 10^{-1}

Exemplificando

Considerando o que acabamos de aprender, o número racional 241,36 – representado em base 10 – pode ser expandido na soma:

$$\begin{aligned} 241,36_{10} &= 2 \times 10^2 + 4 \times 10^1 + 1 \times 10^0 + 3 \times 10^{-1} + 6 \times 10^{-2} = \\ &= 200 + 40 + 1 + 0,3 + 0,06 = 241,36_{10} \end{aligned}$$

Números binários

O sistema de numeração binário é uma outra forma de representar quantidades que se refere a um determinado valor decimal equivalente, mas que o expressa de uma maneira distinta. Podemos observar que ele é mais simples que o sistema decimal, uma vez que usa apenas dois dígitos em vez de dez.

Por outro lado, há um custo para o uso da base binária, que é o aumento do número de dígitos, ou algarismos, para representar uma mesma quantidade ou valor em base decimal.

Assim, é fácil perceber de onde vem o nome da base numérica. Se a base é decimal, significa que se utilizam 10 algarismos. Se for binária, significa que são utilizados dois algarismos distintos para formar a expressão dos valores. E isso pode ser estendido para qualquer base, como a base 17, por exemplo. Acontece que, para nós da tecnologia digital, a teoria matemática sobre base 2 se mostra a mais importante. No dia a dia utilizamos a base decimal, razão pela qual a conversão entre essas duas bases é muito utilizada (PEDRONI, 2010).

A posição de um “1” ou de um “0” em um número binário que indica o seu peso, ou valor dentro do número, assim como a posição de um dígito decimal determinam o valor daquele dígito. Os pesos em um número binário são baseados em potência de 2.

Para aprender a contar no sistema binário, primeiro analise como contamos no sistema decimal. Começamos pelo zero e contamos de forma crescente até nove antes de acabar os dígitos. Então, começamos com o dígito de outra posição (à esquerda) e continuamos a contagem de 10 até 99. Nesse momento, esgotamos todas as combinações de dois dígitos, sendo necessário um terceiro dígito para contar de 100 a 999, e assim sucessivamente.

Uma situação semelhante ocorre quando contamos em binário, com a exção de haver apenas dois dígitos (denominados bits). Começando a contagem: 0, 1. Nesse momento, usamos os dois dígitos, assim incluímos uma nova posição de dígito e continuamos: 10, 11. Esgotamos todas as combinações de dois dígitos, de forma que é necessária uma terceira posição. Com posições para três dígitos podemos continuar a contagem: 100, 101, 110 e 111. Agora precisamos de uma quarta posição de dígito para continuar, e assim por diante.

Conversão de números binários e decimais

Podemos observar que qualquer número decimal pode ser obtido por uma soma de potências de 2 e que existe uma combinação exclusiva de potências de dois para cada número. Por exemplo, o número 13 pode ser obtido pela soma: $8+4+1 = 13$. Isso significa que o número 13 em notação

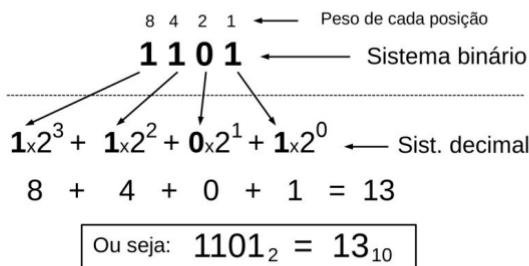
binária é 1101, pois o primeiro bit da direita tem peso $2^0 = 1$, o bit logo à esquerda tem peso $2^1 = 2$ (mas não compõe o número porque o bit vale zero), o outro logo à esquerda vale $2^2 = 4$, e assim por diante. Como somente o segundo bit (da direita para a esquerda) vale zero, então apenas a potência 2 não está presente no somatório. Assim como no sistema decimal, os algarismos que valem zero à esquerda do número não têm importância, razão pela qual 1101 é a mesma coisa que 001101.

É verdade que, principalmente para números maiores, é difícil encontrar quais são as potências de dois a compor o número. No entanto, para o caminho contrário, o processo é mais fácil. Por exemplo, o número em base binária 101 é equivalente a 5 em decimal, pois pode ser descrito pela soma $4+0+1$, ou seja, $101_2 = 1 \times 1^2 + 0 \times 1^1 + 1 \times 1^0 \therefore 4+0+1=5_{10}$

Exemplificando

A Figura 1.7 mostra um exemplo de como um número digital é interpretado ou convertido para a base decimal. Note que podemos usar um número com subíndice para representar a base de outro número, conforme representado no retângulo da ilustração.

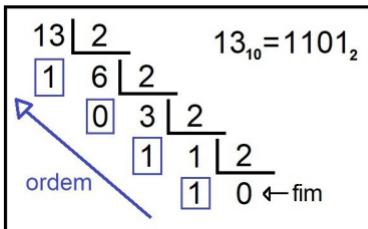
Figura 1.7 | Conversão da base binária para a decimal



Fonte: elaborada pelo autor.

Considerando a conversão de um número na base decimal para binário, apesar de haver meios diferentes, a forma convencional é o método de divisões sucessivas. Nele, inicialmente, o número que se deseja converter é dividido por dois, armazenando-se o resto e o quociente. Este último, então, é novamente dividido por dois, e armazenam-se os resultados, e assim por diante. A ação é repetida até que se atinja zero como quociente. O número binário resultado da conversão será a sequência dos restos das divisões, mas na ordem contrária. Ou seja, o resto da última divisão será o primeiro bit a ser escrito, no lado esquerdo. Vamos ver o exemplo para o número usado, que é o 13 na base decimal.

Figura 1.8 | Método das divisões sucessivas



Fonte: elaborada pelo autor.

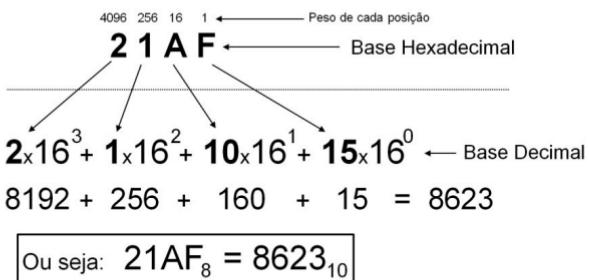
Números hexadecimais

O hexadecimal também é uma base de numeração muito útil para nós, pois, apesar de os bits individuais não aparecerem explicitamente na notação, é uma forma de compactar grandes palavras binárias e facilitar a nossa linguagem.

Assim como o sistema decimal tem dez algarismos, o sistema de numeração hexadecimal tem dezenas de caracteres. Além dos dez dígitos que já usamos no sistema decimal, ele conta com mais 6 dígitos: A, B, C, D, E e F para representar do 10 ao 15, respectivamente.

O sistema hexadecimal é usado principalmente como uma forma compacta de apresentar ou escrever números binários. As conversões entre binário e hexadecimal são relativamente simples, pois a cada 4 bits se forma um único dígito em hexadecimal. Números binários longos são difíceis de serem lidos e escritos porque é fácil omitir ou trocar um bit. Como os computadores entendem apenas 1 s e 0 s, é necessário usar esses dígitos quando se programa em “linguagem de máquina”. Podemos ver na Figura 1.9 um exemplo que converte o número 21AF na base hexadecimal para a decimal.

Figura 1.9 | Conversão da base hexadecimal para a decimal

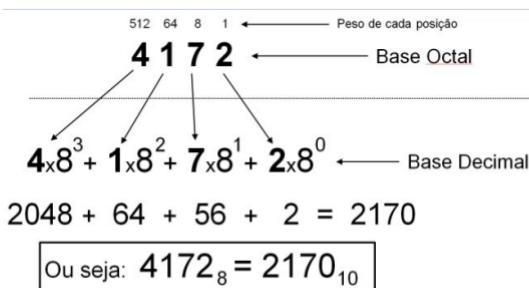


Fonte: elaborada pelo autor.

Números octais

Assim como foi estudado na Figura 1.7, a forma para converter um número em qualquer base para a decimal segue sempre a mesma ideia, embora cada sistema use o seu número de base para gerar os pesos dos algarismos. O número da base para esse sistema, como pressuposto, é 8. Isso significa que todos os números são compostos por apenas 8 algarismos, do 0 ao 7 (TEIXEIRA; TAVARES; PEREIRA, 2017, p. 30), e também que a base da potência referente à posição do algarismo é 8. Assim, o algarismo menos significativo (do lado direito) é multiplicado pelo peso $8^0 = 1$. O algarismo vizinho à esquerda é multiplicado pelo peso $8^1 = 8$. Fazendo esse cálculo para todos os algarismos, e somando os resultados, se obtém o número equivalente representado na base decimal. Vamos ver o exemplo da Figura 1.10, que converte o número 4172 na base octal.

Figura 1.10 – Conversão da base octal para a decimal



Fonte: elaborada pelo autor.

Código BCD

Esta codificação – que significa, em tradução, *decimal codificado em binário* (BCD – *binary coded decimal*) – é, então, uma forma de expressar cada dígito decimal com um código binário. Nesse padrão, cada algarismo em decimal é convertido diretamente em quatro bits, os quais correspondem ao algarismo decimal convertido em binário, mas sem considerar o peso da sua posição.

Por exemplo, o número 547 em decimal será convertido em 3 grupos de 4 bits – o primeiro do grupo (da esquerda para a direita) é o valor 5, em binário; o segundo, o 4; e o terceiro, o 7, ou seja, 0101 0100 0111. Como preferimos ler e escrever em decimal, o código BCD provê uma excelente interface com o sistema binário. Exemplos de tais interfaces são as entradas do teclado e leituras digitais.

Código de Gray

Os bits do código de Gray não têm peso. Não se trata de um código aritmético, ou seja, neste caso não existem pesos associados às posições dos bits.

No método binário convencional, muitas vezes, a alteração de um incremento pode gerar a mudança de mais de um bit na palavra – por exemplo, do número 7 (0111) para o número 8 (1000), quatro bits são alterados. Principalmente para comunicação e transmissão digital, é comum ocorrerem erros de leitura em razão das perturbações ou dos ruídos. Nesse caso, o código de Gray se torna mais eficaz, pois, ao se confundir entre níveis discretos vizinhos, apenas um bit fica errado na transmissão, e a perda de informação tende a ser menor.

EBCDIC

Esta codificação, *Extended Binary Coded Decimal Interchange Code* (ou código de troca decimal codificado em binário estendido), é formada por caracteres 8-bit numéricos e alfanuméricos, e é consequência direta do código BCD com 6-bit. Ele foi criado pela IBM como um padrão no início dos anos 1960 e usado no IBM System/360.

Esse código representa uma primeira tentativa de normalização em paralelo com a normalização ASCII, utilizada pelo governo dos Estados Unidos no final dos anos 1960. No EBCDIC são utilizados pela primeira vez 8-bits, ou 1 *byte*, para codificar um estado, existindo assim a possibilidade de codificar 256 estados diferentes.

ASCII

Este código foi concebido para troca de informações de texto e inicialmente projetado para cobrir o alfabeto americano, com tamanho de 7-bits cada caractere. Posteriormente surgiram os *Extended ASCII Codes*, com 8-bits de largura, para comportar caracteres extras, como acentuações, letras de outro alfabeto, e alguns símbolos.

Os computadores podem entender apenas números, razão pela qual um código ASCII é a representação numérica de um caractere como ‘a’ ou ‘@’ ou uma ação de algum tipo. O ASCII foi desenvolvido há muito tempo e agora os caracteres não imprimíveis raramente são usados para sua finalidade original.

Excesso-3

O código binário em excesso-3 é um código BCD autocomplementar, não ponderado, para expressar expressar números decimais. É um sistema numérico que possui representação com um deslocamento intrínseco, de

valor 3, para ser mais exato. Como fica claro pelo nome, um dígito BCD pode ser convertido no código Excesso-3 correspondente, tarefa para a qual basta adicionar 3. Por exemplo, considere o valor 6, que em BCD vale 0110 – para a codificação Excesso-3, o valor correspondente será $6+3=9$, mas em binário, ou seja: 1001.

O código Excesso-3 foi usado em alguns computadores mais antigos, bem como em caixas registradoras e calculadoras eletrônicas portáteis da década de 1970, entre outros usos.

Caro aluno, esperamos que este primeiro contato tenha sido proveitoso e estimulante. Pode ser que você esteja se perguntando: “o que tudo isso tem a ver com projeto de sistemas digitais?”. Calma: isso ficará bem mais claro já a partir das próximas seções. Você verá que esses conceitos estudados, apesar de não apresentarem uma ligação direta inicialmente, são fundamentais para todo o resto do curso. Não se esqueça de reler com calma as partes em que restaram dúvidas. Se, mesmo assim, não estiver seguro, pergunte o que considera pendente a seu professor ou mesmo faça uma pesquisa na internet. Tente discriminar as páginas mais confiáveis, por nomes que são conhecidos (grandes empresas) ou que são atrelados a algum órgão, como o Instituto de Engenheiros Eletricistas e Eletrônicos do Brasil, por exemplo. Resolva a situação-problema e os exercícios com empenho para consolidar o que foi aprendido e verificar se ainda restam dúvidas.

Bom trabalho!

Sem medo de errar

Na situação-problema desta seção, uma grande granja está criando um novo setor inteiro automatizado para o qual há uma vaga de engenheiro desenvolvedor, que será responsável pelo sistema de fluxo de galinhas no bloco de banho de luz e alimentação. O processo é o seguinte: inicialmente a porta de entrada (PE) está aberta e a porta de saída (OS) está fechada. As galinhas que vêm do pátio de entrada entram uma a uma na sala de ração e são contabilizadas pelo sensor de presença na entrada (SPE), que gera um pulso digital para cada contagem.

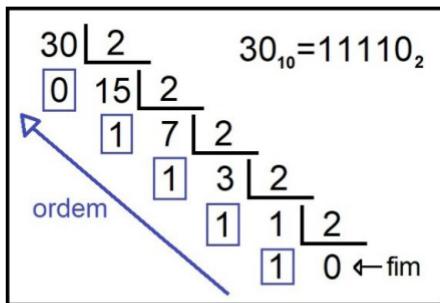
Nessa primeira parte do projeto, será feito o arranjo geral dos blocos lógicos empregados no sistema. O sensor digital de presença será incrementado por um contador, e o valor, correspondente ao número de galinhas, será codificado em BCD para ser impresso em dois displays de 7 seguimentos para indicar o valor aos operadores. O diagrama de blocos lógicos do sistema pode ser visto conforme já apresentado na Figura 1.2.

Sabendo que o número de galinhas que devem entrar em cada sessão de banho de luz e ração é 30 e que um comparador é utilizado para contar as galinhas e fechar a porta de entrada para começar o processo, responda:

- Sabendo que a referência é 30, qual é o valor binário que aparece na contagem quando a porta de entrada (PE) é fechada?
- Qual deve ser o bloco lógico indicado com uma interrogação na figura para que os *displays* possam operar adequadamente?

Para resolver essas questões, basta retornarmos ao conteúdo da seção. Para a resposta da primeira pergunta, a fim de converter o número 30 em binário, devemos usar a técnica de divisões sucessivas, como pode ser visto na Figura 1.11.

Figura 1.11 – Conversão da base decimal para a binária para a SP



Fonte: elaborada pelo autor.

Como resposta da segunda pergunta, o número digital 30 não pode ser diretamente dividido em duas partes, uma com a parte 3 e outra com o algarismo 0, pois seu valor digital não mostra esses dois algarismos separados na notação. Portanto, um codificador/conversor de número binário para BCD deve ser usado no bloco com o ponto de interrogação. Assim, o número digital 11110 que entra nele gera os dois números BCD em sua saída, 0111 e 0000, o que equivale ao número decimal 30. Ainda há a necessidade de outro circuito codificador de BCD para display de 7 segmentos, que pode ser inserido tanto no bloco da interrogação quanto no bloco do display. Assim, cada display pode exibir corretamente o valor contabilizado.

Avançando na prática

Balança digital com resultado impresso em BCD

Uma nova balança digital está sendo desenvolvida pela empresa que você trabalha, e você foi incumbido de projetar a parte de saída de dados e exibição dos valores. O bloco digital que você vai construir recebe o valor fracionário do peso medido com até duas casas decimais, sendo necessário, no caso, que um codificador converta esse valor para o BCD e que, em seguida, um outro codificador ajuste esses novos sinais para serem inseridos propriamente em quatro displays de 7 segmentos, dois para os algarismos decimais antes da vírgula e dois para os depois da vírgula.

Resolução da situação-problema

Devemos usar a mesma ideia da situação-problema anterior para responder este problema, para cuja resolução, a fim de converter o número 30 em binário, devemos usar a maior potência de 2 menor do que 30, que no caso é 16. O número 8, 4 e 2 também podem ser adicionados, completando a soma. Assim, o número 30 na base decimal vale 11110 na base binária.

Faça valer a pena

1. Os números que utilizamos no dia a dia estão na base decimal. Apesar de isso ser algo transparente para muitas pessoas, outras delas ainda não ouviram falar sobre isso ou não sabem o que significa. Os computadores, inicialmente, foram até construídos sobre o sistema numérico de base decimal, mas posteriormente foi constatado que o sistema binário é muito mais eficiente e robusto. Uma vez que esse sistema tem base distinta da que usamos, faz-se necessário entender como são feitas essas representações, bem como se fazem as conversões de valores entre essas bases.

Assinale a alternativa que apresenta corretamente a conversão do número 18 em base decimal para a base binária.

- a. 01001.
- b. 10010.
- c. 10101.
- d. 01110.
- e. 01000.

2. Para sistemas eletrônicos ou computacionais, tanto os dados quanto os sinais que os representam podem ser analógicos ou então digitais (em forma). O termo *dados analógicos* refere-se a informações que são contínuas; os dados digitais, por sua vez, correspondem a informações que possuem estados discretos.

Assinale a alternativa que apresenta corretamente a diferença entre *dados e sinais*.

- a. Os dados são a informação no *hardware*; e sinais, no *software*.
- b. Os dados e os sinais na verdade são exatamente a mesma coisa.
- c. Os sinais podem existir no sistema independentemente dos dados, que nem sempre estão presentes.
- d. Os sinais são a representação física dos dados, que são a informação pura, sem consideração de sua manifestação física.
- e. Os dados são a representação física dos sinais.

3. Todas as informações que circulam na rede e nos computadores devem estar em forma de sequências binárias, pois é essa a linguagem das máquinas mencionadas. No entanto, quando se deseja transmitir ou armazenar sinais da natureza, como imagens, música ou texto falado, estes precisam ser convertidos adequadamente para arquivos binários de computador. Alguns desses padrões de conversão são até populares entre os usuários, como o *mp3*, o *jpeg* ou o *gif*.

Assinale a alternativa que apresenta corretamente o propósito para o qual se aplica o código ASCII.

- a. Transmissão de dados de alta velocidade.
- b. Codificação e compactação de blocos de dados.
- c. Codificação digital de caracteres de texto.
- d. Troca de mensagens de áudio entre aplicativos dedicados.
- e. Monitoramento de tráfego de dados em uma rede.

Seção 2

Introdução à álgebra booleana

Diálogo aberto

Olá, caro aluno. Agora que você já teve os primeiros contatos com a teoria mais básica dos sistemas digitais, vamos começar nosso estudo sobre projetos de circuitos digitais. Para entender como os circuitos lógicos são projetados, devemos saber o que é uma tabela-verdade e o que ela representa para um bloco digital. Você verá como uma tabela-verdade deve ser elaborada corretamente para prover solução em uma aplicação que geralmente envolve controle digital. Além disso, aprenderá também como os circuitos digitais podem ser simplificados, gerando um circuito equivalente mais econômico. Por fim, vamos ver não só como implementar circuitos lógicos com base na expressão booleana, mas também como obter esta mediante um circuito lógico.

Esses conceitos são fundamentais para o engenheiro projetar circuitos lógicos, ou mesmo para aplicações industriais de mais alto nível. Assim como na seção passada, vamos tratar de um problema que envolva o assunto abordado nessa seção e que tenha grande proximidade com problemas reais encontrados no trabalho cotidiano de um engenheiro de projetos. O problema agora envolve o mesmo contexto da situação anterior: a automação de uma granja. No entanto, sua tarefa agora é outra bem diferente. Você, como gerente de projetos de automação de uma grande granja de galinhas, tem como desafio desenvolver um simples sistema digital para acionar a iluminação do galpão da granja, que é representada por uma única saída (uma lâmpada) e que pode ser controlada por dois interruptores, um em cada entrada da granja. Uma chave-disjuntor deve ser usada para habilitar (ou não) todo o sistema. Portanto, o sistema deve ter 3 entradas digitais: o interruptor 1 (A), o interruptor 2 (B) e a chave-disjuntor (C), cada uma das quais representa 1 bit apenas. A saída do sistema é uma só, a lâmpada (S), que representa o sistema de iluminação como um todo e que é sempre ligada ou desligada conjuntamente. Note que os dois interruptores não devem funcionar no clássico padrão “3-way”, com o qual a mudança de estado de qualquer um deles sempre ocasiona a mudança na saída. A luz deve ser ligada ($S=1$) se qualquer um dos interruptores estiver ativo. O sistema só deve atuar se o disjuntor estiver ativo ($C=1$). As quatro etapas do projeto a que você deve atender são:

- a. Circuito lógico com uma porta AND e uma porta OR.
- b. Tabela-verdade para o seu circuito digital.
- c. Expressão para a função de transferência booleana mediante método dos mintermos.
- d. Circuito lógico simplificado pelo método “Mapa de Karnaugh” (deve bater com a resposta da letra “a”).

Olhando à primeira vista assim, pode parecer um trabalho bem difícil, não é? Mas fique tranquilo, pois, ao final desta seção, você verá que, em 10 ou 20 minutos, conseguirá responder tudo sem qualquer problema.

Vamos começar?

Não pode faltar

Neste capítulo vamos estudar as bases matemáticas da lógica binária fundamentais para a construção de circuitos e sistemas digitais, cujo nome é *álgebra booleana*.

De forma intuitiva e natural, usamos os termos lógicos no dia a dia mesmo sem perceber. Por exemplo, se uma namorada fala para o companheiro “querido, vamos ao cinema amanhã se não chover e se você estiver a fim”, podemos ver a frase de forma lógica, na qual a saída (resultado/consequência), que é a variável binária “ida ao cinema”, será verdadeira somente se as duas condições, também binárias, forem satisfeitas: 1^a: não chover; 2^a: o companheiro querer ir.

Pensando nesses termos e trazendo para um contexto geral, em 1850, o matemático e logicista irlandês George Boole desenvolveu um sistema matemático para formulação de declarações lógicas com símbolos, de modo que pudessem ser resolvidos de forma similar à álgebra comum de Floyd (2007).

A álgebra booleana tem sido fundamental no desenvolvimento da eletrônica digital, sendo encontrada em todas as linguagens de programação modernas. Também pode ser encontrada na teoria dos conjuntos e estatística (GIVANT; HALMOS, 2009).

Na década de 1930, enquanto estudava circuitos de comutação, Claude Shannon observou que também se poderia aplicar as regras da álgebra de Boole nesse cenário, introduzindo a álgebra de comutação como uma maneira de analisar e projetar circuitos por meios algébricos em termos de portas lógicas. Shannon já tinha à disposição o aparato matemático abstrato, quando, então, lançou sua álgebra de comutação como a álgebra booleana de dois elementos.

O termo *lógica* é aplicado a circuitos digitais usados para implementar as funções lógicas. Diversos tipos de circuitos lógicos digitais são os elementos básicos que formam os blocos construtivos de sistemas digitais complexos como o computador.

Algumas das funções lógicas comuns são: “a comparação, a aritmética, a conversão de códigos, a codificação, a decodificação, a seleção de dados, o armazenamento e a contagem” (TOCCI; WIDMER; MOSS, 2011, p. 101).

Introdução e operações com álgebra booleana

As operações e os circuitos lógicos são feitos sobre sinais lógicos (ou booleanos). “Todos esses sinais são tratados como **variáveis ou constantes booleanas.**” (TEIXEIRA; TAVARES; PEREIRA, 2017, p. 57) Uma variável digital pode assumir valores compostos, razão pela qual gera um vetor de bits. No entanto, isso deve ser informado no contexto da abordagem. Inicialmente, todas as variáveis tratadas aqui serão booleanas, ou seja, de apenas 1 bit de tamanho. As primeiras operações que vamos estudar recebem o mesmo nome usado para os circuitos.

Operações lógicas

Apesar das incontáveis possibilidades de combinações, existem somente três operações lógicas elementares: a **NOT** (NÃO), a **OR** (OU) e a **AND** (E). Cada uma dessas operações lógicas primordiais possui uma porta lógica (símbolo gráfico) correspondente, que fisicamente tem o mesmo comportamento que o seu modelo matemático binário.

Refletá

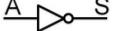
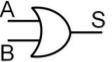
As portas lógicas são os elementos essenciais para a construção de sistemas digitais, que são gerados pela associação de muitas delas, geralmente aos milhares ou até em milhões de unidades.

Um profissional dessa área deve ser capaz de escolher as portas apropriadas, bem como as suas conexões, para prover uma solução útil a uma dada aplicação de controle digital.

Além das três portas elementares, há uma outra que também é muito importante, mas que não é elementar, pois é formada por um pequeno arranjo delas. A operação lógica **XOR – Exclusive OR** (OU Exclusivo) recebe esse nome próprio e também um símbolo próprio. O comportamento dessa porta é que sua saída gere valor verdadeiro somente se apenas uma das entradas for verdadeira.

A Figura 1.12 ilustra as portas lógicas e seus respectivos funcionamentos. Note que, no lado esquerdo das tabelas, são apresentadas as suas entradas, que podem ser uma ou várias (A , B , C ... etc.) e o lado direito apresenta sempre a sua única saída (S).

Figura 1.12 | Portas lógicas

Porta NOT <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><th>A</th><th>S</th></tr> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </table>  $S = \bar{A}$	A	S	0	1	1	0	Porta OR <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><th>A</th><th>B</th><th>S</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>  $S = A + B$	A	B	S	0	0	0	0	1	1	1	0	1	1	1	1									
A	S																														
0	1																														
1	0																														
A	B	S																													
0	0	0																													
0	1	1																													
1	0	1																													
1	1	1																													
Porta AND <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><th>A</th><th>B</th><th>S</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>  $S = A \cdot B$	A	B	S	0	0	0	0	1	0	1	0	0	1	1	1	Porta XOR <table border="1" style="margin-left: auto; margin-right: auto;"> <tr><th>A</th><th>B</th><th>S</th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>  $S = A \oplus B$	A	B	S	0	0	0	0	1	1	1	0	1	1	1	0
A	B	S																													
0	0	0																													
0	1	0																													
1	0	0																													
1	1	1																													
A	B	S																													
0	0	0																													
0	1	1																													
1	0	1																													
1	1	0																													

Fonte: elaborada pelo autor.

Como é possível perceber, as operações das portas OR, AND e XOR devem ser feitas sobre dois ou mais sinais (mesmo que sejam constantes) booleanos. No outro extremo, a operação NOT é feita sempre sobre somente um único sinal binário, mesmo que este seja resultado de uma outra função lógica.

- **Porta inversora, ou NOT:** o inversor (circuito NOT) realiza a operação denominada inversão ou complementação. O inversor troca um nível lógico para o nível lógico oposto. Em termos de bit, ele troca 1 por 0 ou 0 por 1. Esta é a única porta que é aplicada a um único sinal digital, e só pode ser usada desta forma: individualmente. Na Figura 1.12 podemos ver o símbolo, o comportamento e a notação para expressão matemática booleana da porta inversora. A porta NOT é um correspondente análogo a um circuito formado por uma chave NF que, por exemplo, poderia acionar uma lâmpada.
- **Porta AND:** uma porta AND pode ter duas ou mais entradas e realizar uma operação conhecida como “multiplicação lógica”. Uma porta AND produz uma saída de nível ALTO apenas quando todas as entradas forem nível ALTO. Quando qualquer uma das entradas for nível BAIXO, a saída será nível BAIXO. Portanto, o propósito básico da porta AND é determinar quando certas condições são simultaneamente verdadeiras, conforme indicado pelos níveis ALTOS em todas

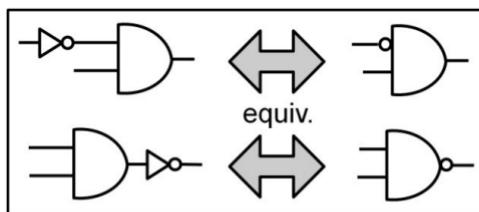
as entradas para produzir um nível ALTO na saída, indicando que todas essas condições são verdadeiras. A porta AND é análoga a um circuito com duas chaves em série que acionam uma lâmpada.

- **Porta OR:** uma porta OR pode ter duas ou mais entradas e realiza o que conhecemos como “adição lógica”. Uma porta OR produz um nível ALTO na saída quando qualquer das entradas for nível ALTO. Assim, a saída será nível BAIXO apenas quando todas as entradas estiverem em nível BAIXO. Portanto, uma porta OR determina quando uma ou mais de suas entradas estiverem em nível ALTO e produz um nível ALTO na sua saída para indicar essa condição. “A porta OR corresponde analogamente a um circuito com duas chaves paralelas, que acionam uma lâmpada.” (TEIXEIRA; TAVARES; PEREIRA, 2017, p. 61)
- **Porta XOR:** a saída de uma porta OU Exclusivo é nível ALTO apenas quando somente uma de suas entradas é ALTO. Caso contrário, sua saída é nível BAIXO, conforme é ilustrado na Figura 1.12.

Há também algumas outras portas compostas que recebem um nome próprio, como a **NAND** ou **NOR**, que nada mais são que portas AND e OR seguidas de uma NOT nas suas respectivas saídas.

Para simplificar a notação de portas compostas com a NOT, usaremos a equivalência apresentada na Figura 1.13 – note que a segunda porta composta se trata da NAND.

Figura 1.13 | Equivalências da porta NOT para simplificação



Fonte: elaborada pelo autor

As leis e regras da álgebra booleana

Assim como a álgebra tradicional que aprendemos no ensino fundamental, a álgebra booleana também possui as próprias regras (PEDRONI, 2010). Podemos afirmar que as leis são exatamente as mesmas e não apresentam nenhuma novidade ou adaptação. Para as regras, as nove primeiras são totalmente intuitivas e lógicas, usando o próprio nome como adjetivo. As demais

podem ser encontradas com base na aplicação das leis apresentadas e algumas regras iniciais, conforme pode ser visto no Quadro 1.1 a seguir:

Quadro 1.1 | Regras e leis booleanas

Leis básicas da álgebra booleana		
Comutativa:	$A + B = B + A$ e $A \cdot B = B \cdot A$	
Associativa:	$A + (B + C) = (A + B) + C$ e $A \cdot (B \cdot C) = (A \cdot B) \cdot C$	
Distributiva:	$A + (B + C) = A \cdot B + A \cdot C$	
Regras da álgebra booleana		
1: $A + 0 = A$	5: $A + A = A$	9: $\overline{\overline{A}} = A$
2: $A + 1 = 1$	6: $A + \overline{A} = 1$	10: $A + A \cdot B = A$
3: $A \cdot 0 = 0$	7: $A \cdot A = A$	11: $A + \overline{A} \cdot B = A + B$
4: $A \cdot 1 = A$	8: $A \cdot \overline{A} = 0$	12: $(A + B) \cdot (A + C) = A + B \cdot C$

Fonte: elaborado pelo autor

Note que a lei distributiva possui o seu caminho inverso, que corresponde a nosso famoso “põe em evidência”. Se observarmos a terceira lei, da direita para a esquerda, podemos ver que a variável “A” pode ser posta em evidência no lado direito, retornando a expressão do lado esquerdo. Atente para essa passagem, pois será muito útil logo à frente, em simplificação lógica.

Operações aritméticas booleanas

Estas operações também seguem os mesmos princípios aplicados à álgebra tradicional, embora feitas sobre variáveis booleanas. Vamos estudar aritmética com valores digitais compostos na próxima seção. Agora usaremos apenas variáveis e constantes booleanas, ou seja, aquelas com apenas 1 bit. Na Figura 1.14, podemos ver as operações aritméticas básicas.

Figura 1.14 | Operações aritméticas booleanas

Adição Booleana	Multip. Booleana	Complemento
$0 + 0 = 0$	$0 \cdot 0 = 0$	
$0 + 1 = 1$	$0 \cdot 1 = 0$	
$1 + 0 = 1$	$1 \cdot 0 = 0$	
$1 + 1 = 1$	$1 \cdot 1 = 1$	
		$\overline{0} = 1$
		$\overline{1} = 0$

Fonte: elaborada pelo autor.

Assimile

Quando estivermos tratando de variáveis ou constantes booleanas, as operações aritméticas “MAIS” e “VEZES” são diretamente correspondentes às operações lógicas OR e AND, respectivamente.

Para uma expressão booleana múltipla ou composta, ou seja, com mais de uma operação lógica, deve estar muito claro qual é a ordem que se deve seguir entre as operações.

Se estiver sobre uma única variável booleana, a operação NOT (barrado) deve ser a primeira a ser realizada (PEREIRA; TEIXEIRA, 2018, p. 90). Em seguida, o que estiver entre parênteses. Depois as operações AND (ou multiplicação), e por fim as somas (ou operações OR).

Refita

De maneira geral, podemos entender a ordem das operações binárias da mesma maneira que sequenciamos as operações com base decimal.

Blocos lógicos combinacionais e tabela-verdade

Qualquer bloco lógico ou circuito digital pode ser interpretado como uma função de transferência, igualmente digital, para cada uma das saídas do sistema individualmente. Assim, o comportamento lógico de cada saída do sistema em questão pode ser descrito por uma expressão booleana. Se fizermos uma tabela com todas as combinações possíveis para as entradas e encontrarmos o valor que a saída assume, (0 ou 1) para cada uma dessas combinações, chegamos ao mapeamento completo da função de transferência digital do circuito. Essa tabela é chamada de **tabela-verdade** (PEDRONI, 2010). Ou seja, as tabelas que estão representadas nas figuras 1.12, 1.15 e 1.16, são tabelas-verdade, mesmo que para poucas entradas ou somente uma.

Como veremos logo adiante, para uma mesma dada tabela não existe somente uma única expressão lógica correspondente (ou combinação de portas), ou seja, que apresente o mesmo comportamento descrito para a tabela-verdade. Veremos que diferentes expressões e circuitos podem apresentar uma mesma tabela-verdade correspondente.

Método dos mintermos

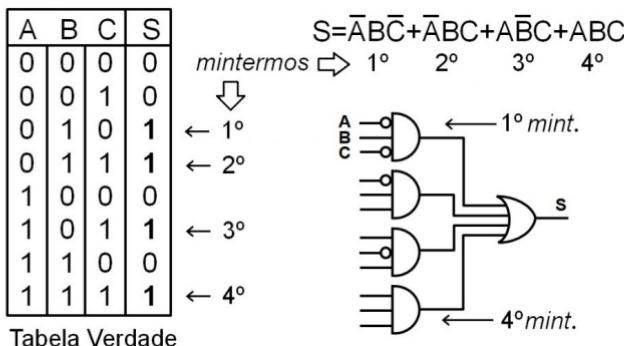
A forma mais tradicional e direta de obter um circuito mediante uma tabela-verdade é usando *mintermos*, ou seja, fazendo uso do método dos *mintermos*. A aplicação desse método funciona da seguinte forma:

1. Cada saída verdadeira (igual a 1) da tabela-verdade corresponde a um *mintermo*.
2. Cada *mintermo* é formado pela operação AND entre **todas** as entradas do circuito.
3. As entradas que valem ZERO para cada *mintermo* ficam barradas nele.
4. A saída é o resultado de uma operação OR entre todos os *mintermos*.

Podemos dizer que esse método é uma soma de produtos, que são os *mintermos* (IODETA; CAPUANO, 1997). Vamos deixar tudo isso mais claro com o exemplo da Figura 1.15, com uso de uma tabela-verdade qualquer a expressão da sua função booleana por *mintermos*, e o seu circuito lógico correspondente.

Figura 1.15 | Função booleana por *mintermos*

Função booleana por *mintermos*



Fonte: elaborada pelo autor.

Perceba que cada um dos *mintermos* (4 nesse caso) gera uma das portas AND do circuito.

Método dos *maxtermos*

Funciona de maneira inversa aos *mintermos*. Ao passo que nos *mintermos* a saída é uma soma dos produtos, nos *maxtermos* ela é um produto de somas. A aplicação desse método funciona da seguinte forma:

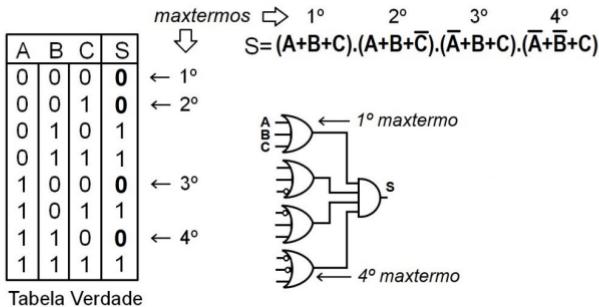
1. Cada saída falsa (igual a 0) da tabela-verdade corresponde a um *maxtermo*.
2. Cada *maxtermo* é formado pela operação OR entre **todas** as entradas do circuito.

3. As entradas que valem UM para cada *mintermo* ficam barradas nele.
4. A saída é o resultado de uma operação AND entre todos os *maxtermos*.

Vamos utilizar novamente a tabela da Figura 1.15 para mostrar como um circuito equivalente pode ser gerado por *maxtermos*, visto na Figura 1.16.

Figura 1.16 | Função booleana por *maxtermos*

Função booleana por *maxtermos*



Fonte: elaborada pelo autor.

Assimile

Algumas das funções lógicas (ou blocos lógicos) mais comuns são: comparação, contagem, aritmética, codificação, seleção de dados, armazenamento e controle.

Simplificação booleana

Como já foi dito, uma única tabela-verdade permite compatibilidade entre várias expressões booleanas que, embora sejam distintas, são equivalentes por apresentarem o mesmo comportamento de função de transferência. “Um dos grandes objetivos em projetos de computação é, além de prover a tabela-verdade apropriada para a aplicação, permitir reduzir o circuito lógico ao menor possível, gerando economia na construção dele.” (TOCCI; WIDMER; MOSS, 2011, p. 103)

“As simplificações são feitas pela aplicação direta das leis e regras da álgebra booleana”. (PEREIRA, 2018, p. 85) Considerando o caso da expressão obtida por mintermos, que é sempre um caminho fácil e direto, o que se utiliza é basicamente a 3ª lei (de forma invertida) e as regras 6 e 4 da sequência.

Exemplificando

Aproveitando o caso da tabela-verdade apresentada na Figura 1.15, podemos ainda simplificá-la da seguinte forma:

Figura 1.17 | Simplificação booleana

Simplificação booleana

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

Tabela Verdade

$$S = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}C + ABC$$

Colocando os termos $A\bar{B}$
e AC em evidência:

$$S = \bar{A}B(\bar{C} + C) + AC(\bar{B} + B)$$

E usando as regras 6 e 4:

$$S = \bar{A}B \cdot 1 + AC \cdot 1$$

$$S = \bar{A}B + AC$$

Fonte: elaborada pelo autor.

Mapa de Karnaugh

Esta técnica provê um método sistemático para simplificação de expressões booleanas. Como plano de fundo, são realizadas exatamente as etapas de simplificação mostradas, mas de uma forma indireta, por meio do arranjo e das regras que são o núcleo do método. Como resultado dessa simplificação pelo mapa é encontrada a expressão mais simplificada possível para o caso, conhecida como “expressão mínima”.

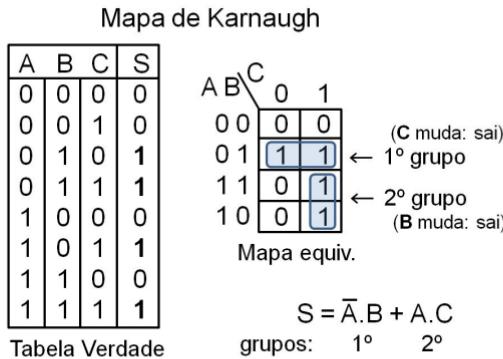
Um mapa de Karnaugh é similar a uma tabela-verdade porque todos os valores possíveis das variáveis de entrada e a saída resultante, para cada valor, estão presentes nesse mapa. Em vez de estar organizado em colunas e linhas como uma tabela-verdade, o mapa de Karnaugh é um arranjo de células, no qual cada uma destas representa um valor binário das variáveis de entrada. Os passos para aplicar esse método são os seguintes:

1. Monta-se um quadro como uma tabela, com as variáveis de entrada divididas em dois grupos: um para as linhas da tabela e outro para as colunas dela.
2. Em seguida, mapeiam-se todas as combinações possíveis para as entradas binárias, seguindo a ordem do “código de Gray”, e não a sequência binária comum.

- A tabela é então preenchida de acordo com os valores da saída da tabela-verdade utilizada.
- Criam-se os grupos, um dos quais pode ser formado por 2 quadros (pares), 4 (quartetos), 8 (octetos), agrupamentos de 16 bits, etc. – sempre com suas saídas iguais a “um”, consecutivas.
- Essa formação do grupo também pode ser quadrada ou retangular, desde que com essas quantidades e com o maior número de elementos possível.
- Uma saída 1 pertencente a um grupo pode ser usada para formar outro, mas se possível deve ser evitado.
- Todas as saídas unitárias devem ter sido agrupadas no fim do processo.
- Se uma saída 1 está isolada de qualquer outra, ela forma um grupo de apenas 1 elemento.

Vamos aplicar agora para o nosso exemplo da seção:

Figura 1.18 | Aplicação do Mapa de Karnaugh



Fonte: elaborada pelo autor

Devemos atentar para uma característica importante desse método, que é a “circularidade” do mapa. Isso significa que o elemento mais abaixo de uma coluna, ou seja, o último, pode ser agrupado com o localizado mais acima, ou seja, o primeiro, casos os dois sejam verdadeiros. Podemos interpretar isso como o nosso *mapa-múndi*, que, apesar de ser um retângulo, sabemos que representa a Terra, que é um globo (geoide ou “quase” esfera). Isso vale para elementos verdadeiros nos dois extremos de uma mesma linha. O caso mais extremo ocorre quando os “4 cantos” do mapa de *karnaugh* são verdadeiros, formando um grupo que se conecta pelas “costas” do mapa circular.

Vale lembrar que, ainda para esse caso, se outras saídas verdadeiras estiverem vizinhas aos cantos e permitirem a geometria retangular dos grupos, esse grupo extremo, como qualquer outro, circular ou não, pode ser ampliado para otimizar a simplificação da seção.

Sem medo de errar

Agora que você terminou de acompanhar o conteúdo mais teórico da seção, vamos colocar em prática os ensinamentos para consolidação do estudo? Assim como na seção anterior, vamos tratar de um problema que envolva o assunto abordado aqui nesta seção e que tenha uma grande proximidade com problemas reais, encontrados no trabalho cotidiano de um engenheiro de projetos. O problema agora ainda envolve o contexto da situação anterior: a automação de uma granja. No entanto, a tarefa agora é outra bem diferente.

Você, como gerente de projetos de automação da rede de granjas, agora tem como objetivo desenvolver um simples sistema digital para acionar a iluminação do galpão da granja, que é representada por uma única saída (uma lâmpada) e que pode ser controlada por dois interruptores, um em cada entrada da granja. Uma chave-disjuntor deve ser usada para habilitar ou não o sistema. Portanto, o sistema digital deve ter 3 entradas digitais: o interruptor 1 (A); o interruptor 2 (B); e a chave-disjuntor (C) –; todas de 1 bit apenas. A saída do sistema é uma só: a lâmpada (S), que representa o sistema de iluminação como um todo. Note que os dois interruptores não devem funcionar no clássico padrão “3-way”, com base no qual a mudança de estado de qualquer um deles sempre ocasiona a mudança na saída. A luz deve ser ligada ($S=1$) se qualquer um dos interruptores estiver ativo. O seu sistema só deve atuar se o disjuntor estiver ativo ($C=1$). As etapas do projeto que você deve atender são: a) circuito lógico com uma porta AND e uma porta OR; b) tabela-verdade para o circuito digital; c) expressão para a função de transferência booleana com base no método dos mintermos; d) circuito lógico simplificado por meio do método “mapa de Karnaugh” (deve bater com a resposta da letra “a”).

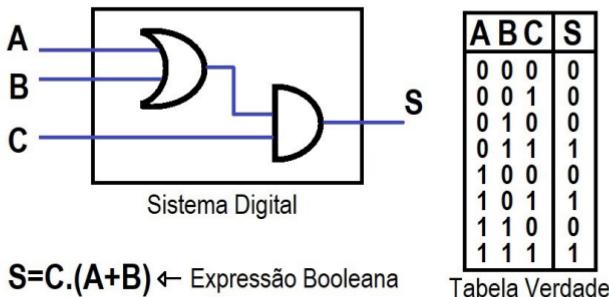
Resolução:

- Como os dois interruptores podem ligar a luz de forma paralela e independente, então os dois sinais devem entrar em uma porta OR (OU). Uma forma clara de perceber isso é pela frase: “a luz deve ser ligada se for acionado o interruptor 1 ou o interruptor 2”. No entanto,

a saída desta porta (OU) não deve estar diretamente conectada à saída, pois a luz só pode ser acionada pelos interruptores se o interruptor geral estiver habilitado. Sabemos que a porta que permite que uma variável binária seja habilitada é a AND (E) entre o sinal da variável (que é a soma $A+B$) e o sinal “habilitador” (que neste caso é o disjuntor). Ou seja, $S=C.(A+B)$. Pois se o sinal “habilitador” for 1, a saída será $S=1.(A+B)=A+B$. Ou seja, a saída é acionada se algum interruptor for acionado. No entanto, se o sinal habilitador for zero, a saída será $S=0.(A+B)=0$. Ou seja, sempre desligada.

- b. Podemos montar a tabela-verdade com base na lógica do sistema. A luz (S) deve ser ligada se o interruptor 1 (A) estiver acionado ou se o interruptor 2 (B) estiver acionado. Isso nos traz a ideia da operação: $S=A+B$. No entanto, a chave-disjuntor geral (C) é usada para habilitar o sistema, razão pela qual a saída fica $S=C.(A+B)$, como vimos. Portanto, a saída deve ser verdadeira apenas nas linhas (condições) da tabela-verdade em que C é verdadeiro, e pelo menos um dos interruptores (A ou B) é também verdadeiro.

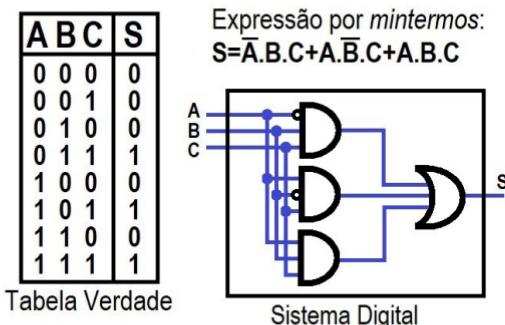
Figura 1.19 – Função de transferência do sistema digital



Fonte: elaborada pelo autor.

- c. Aqui basta aplicar a técnica dos *mintermos*. Nela, cada *mintermo* corresponde a uma saída verdadeira. Todos os *mintermos* são formados por uma operação AND entre todas as entradas. E as entradas que valem ZERO para o *mintermo* entram “barradas” (invertidas ou negadas). A saída é o resultado de uma operação OR entre todos os mintermos.

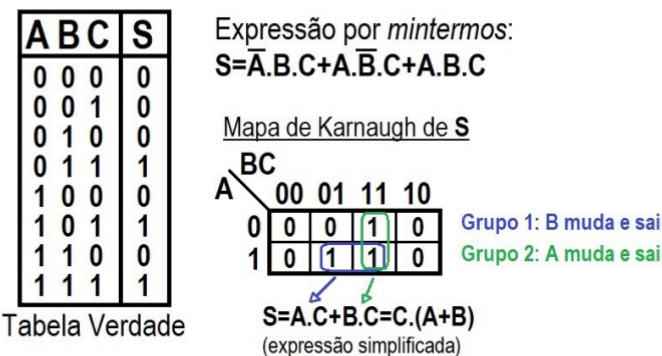
Figura 1.20 – Expressão pelo método dos *mintermos*



Fonte: elaborada pelo autor.

- d. Aqui devemos aplicar a técnica do mapa de *Karnaugh*. Lembre-se: o mapa para 3 entradas pode ter duas colunas e uma linha ou vice-versa, e, independentemente disso, o resultado será o mesmo.

Figura 1.21 – Simplificação pelo método mapa de *Karnaugh*



Faça valer a pena

1. Qualquer bloco lógico ou circuito digital pode ser interpretado como uma função de transferência, igualmente digital, para cada uma das saídas do sistema individualmente. Assim, o comportamento lógico de cada saída do sistema em questão pode ser descrito por uma expressão booleana. Se fizermos uma tabela com todas as combinações possíveis para as entradas e encontrarmos o valor que a saída assume, (0 ou 1), para cada uma dessas combinações, temos o mapeamento completo da função de transferência digital do circuito. Essa tabela é chamada de tabela-verdade.

Na teoria de eletrônica digital, a técnica conhecida como “mapa de Karnaugh” é utilizada para:

- a. obter a tabela-verdade pela função de transferência booleana do circuito.
- b. ajustar a expressão booleana para que se utilizem apenas portas AND e NOT no circuito.
- c. encontrar quais são as entradas do sistema que estão relacionadas com uma específica saída e quais entradas são independentes para a ativação da saída.
- d. obter a expressão lógica equivalente mais simplificada possível para a tabela-verdade utilizada.
- e. gerar todas as equações booleanas que sejam correspondentes à tabela-verdade utilizada.

2. No desenvolvimento de projetos de circuitos e sistemas digitais, a modelagem digital é feita toda mediante álgebra booleana, razão pela qual esse conhecimento se mostra fundamental para o engenheiro em eletrônica digital.

Sabendo que as funções de transferência digitais são dadas por uma expressão lógica ou booleana, considere a seguinte expressão para um bloco lógico:

$$S = A \cdot B + \overline{A} \cdot C \cdot \overline{B} + \overline{A}$$

Para o caso apresentado, qual das seguintes afirmativas pode ser considerada *incorrecta*?

- a. a expressão pode ser simplificada para uma outra reduzida, porém equivalente.
- b. a expressão está formada pelo padrão de *mintermos*.
- c. a expressão deve ser classificada como composta, pois utiliza múltiplas operações lógicas.
- d. o circuito equivalente pode ser gerado por 3 portas fundamentais: NOT, OR e AND.
- e. seguindo o princípio usado na base decimal, as operações AND (multiplicação) devem ser feitas antes das OR (ou soma).

3. Para um bloco lógico de um circuito digital, uma única tabela-verdade permite compatibilidade entre várias expressões booleanas distintas, mas que são equivalentes, por apresentarem o mesmo comportamento de função de transferência. No entanto, cada expressão diferente corresponde a um

circuito diferente, que pode possuir um maior ou menor número de portas lógicas na sua construção.

Considerando a função de transferência booleana com a seguinte expressão lógica: $S = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$

qual das alternativas a seguir representa a nova função equivalente gerada pela aplicação do mapa de Karnaugh?

- a. $S = \overline{B} \cdot C + \overline{A} \cdot \overline{C} + A \cdot \overline{B} \cdot C$
- b. $S = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$
- c. $S = \overline{A} \cdot \overline{B} + B \cdot \overline{C} + A \cdot B$
- d. $S = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B + A \cdot \overline{B} + A \cdot \overline{C}$
- e. $S = \overline{B} \cdot C + B \cdot \overline{C}$

Seção 3

Circuitos digitais

Diálogo aberto

Prezado aluno! Em nosso dia a dia somos rodeados de circunstâncias em cujas ocorrências somente dois estados são possíveis: por exemplo, luz apagada ou luz acesa, porta fechada ou porta aberta, ligado ou desligado, vazio ou cheio, alto ou baixo, ou então, mais precisamente, verdadeiro ou falso. Essa lei é conhecida como lógica booleana. Tal como na álgebra comum, em que usamos símbolos como X ou Y para representar valores numéricos, na álgebra booleana empregamos letras como A e B para representar dois valores possíveis: verdadeiro ou falso.

Em 1854, o matemático George Boole, como vimos, descreveu um conjunto de regras (denominado de álgebra booleana) capaz de relacionar essas circunstâncias (as entradas) a fim de permitir uma tomada de decisão (a saída). A proposta desta seção é estudarmos as expressões lógicas correspondentes a esse conjunto de regras, as simplificações e as conversões em portas e circuitos lógicos.

Segundo Bollmann (1997), existem inúmeras vantagens em se aplicar a álgebra booleana, sobretudo na composição de sistemas de controle e segurança de prensas pneumáticas (ou eventualmente hidráulicas) amplamente utilizadas na indústria.

Com base nesse cenário, o desafio desta seção é elaborar um sistema digital que permita oferecer segurança na operação de uma prensa pneumática. Como sabemos, as prensas, em geral, são extremamente perigosas, pois são muito utilizadas para comprimir objetos com pressões que variam de alguns quilos até várias toneladas, as quais estão, por isso, associadas a graves acidentes de trabalho.

Portanto, com base nisso, o desafio agora será projetar um sistema que garanta a segurança na operação de uma prensa pneumática que só poderá ser ligada quando os dois botões de acionamento mais o pedal de segurança forem colocados em ação simultaneamente, contexto no qual uma grade de proteção deverá estar abaixada para, então, garantir a operação e a segurança do operador.

Para esse projeto, você deverá elaborar uma tabela de correspondências entre as variáveis de entrada (botões de acionamento) e a variável de saída, que serão a alimentação e o acionamento da prensa.

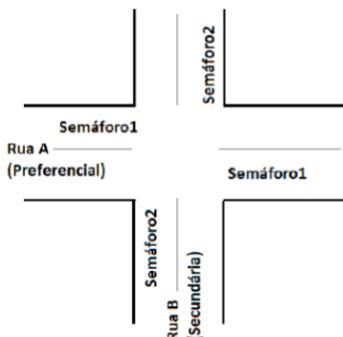
Uma dica preciosa, para isso, é elaborar inicialmente uma tabela-verdade com o teste de todas essas variáveis e, na sequência, o projeto final com as portas lógicas mostrando o circuito.

Podemos utilizar um circuito lógico para solucionar problemas quando acontecerem determinadas situações, representadas pelas variáveis de entrada. Para construirmos esses circuitos, inicialmente partimos das expressões características deles, que são obtidas pelas respectivas tabelas-verdade. Nesse sentido, vamos mostrar a seguir como obter uma tabela-verdade para resolver um problema com base em uma situação prática.

Cabe destacar que o projeto apresentado, embora simule situação real, é didático e serve para descrever o método de realização, podendo ser conduzido na prática como modelo para a solução de pequenos problemas ou, ainda, para a construção de circuitos periféricos dentro de sistemas digitais.

A Figura 1.22 ilustra um cruzamento de ruas com sinalização semafórica.

Figura 1.22 | Cruzamento de vias com sinalização



Fonte: elaborada pelo autor.

Conforme podemos ver na Figura 1.22, temos um cruzamento das ruas A e B, onde queremos instalar um sistema automático para os semáforos, com as seguintes características:

1. Quando houver carros transitando somente na Rua B, o semáforo 2 deverá permanecer verde para que esses veículos possam trafegar livremente.
2. Quando houver carros transitando somente na Rua A, o semáforo 1 dever permanecer verde pelo mesmo motivo.
3. Quando houver carros transitando nas Ruas A e B, é preciso abrir o semáforo para a Rua A, pois esta é a preferencial.

4. Haverá apenas as cores *vermelho* e *verde* nos semáforos.

Com base nas considerações anteriores, elabore a tabela-verdade conforme as convenções apresentadas.

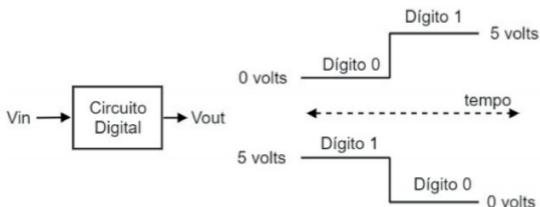
Gostou desse desafio? Continue se preparando e aprendendo mais, já que existe uma enormidade de aplicações para a álgebra booleana e seus circuitos lógicos.

Não pode faltar

Conceito nível lógico

A maneira como um circuito digital entrega um resultado na saída em função das entradas pode ser chamada de “*lógica do circuito*”, segundo a qual cada circuito digital segue um procedimento específico, ou seja, a própria lógica. Os circuitos lógicos são desenvolvidos (projetados) para gerar tensões de saída nas faixas entre o VCC (alimentação) e GND (terra de sinal ou referência 0V). Entre esses limites de valores haverá sempre VOH (um valor de tensão que representa nível lógico alto) e VOL (um valor de tensão que representa nível lógico baixo), conforme pode ser visto pela Figura 1.23.

Figura 1.23 | Níveis lógicos de tensão



Fonte: adaptada de Tocci, Widmer e Moss (2003).

Na Figura 1.23 há um circuito digital com entrada *Vin* e sua consequente saída *Vout* e as duas possíveis formas de onda, tanto para o sinal na entrada quanto para a resposta na saída. No circuito mencionado existem funções lógicas que podem representar apenas dois estados: ligado ou desligado; zero ou um; aberto ou fechado. Portanto, concluímos que tanto na entrada quanto na saída desses blocos são permitidos somente esses dois “estados”.

Agora, daremos um passo maior. Vamos combinar o uso dessas portas para formar circuitos mais complexos, ou seja, circuitos lógicos combinaçãois. Tenha em mente que utilizaremos circuitos com funções lógicas

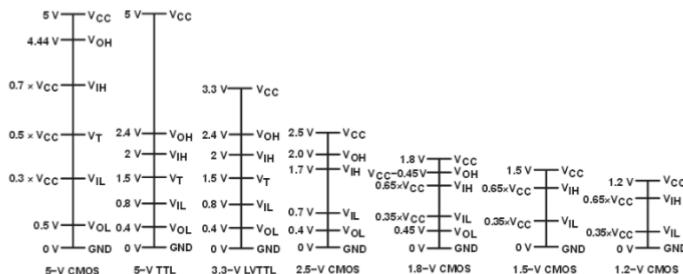
fixas, como os circuitos integrados a seguir: 74XX283; 74XX138; 74XX148; 74XX154; 74XX47; 74XX151; 74XX139; 74XX280; 74XX85; 74XX147; 74XX157 e 74XX42, para exemplificar nossos problemas. Alguns desses circuitos, por exemplo, são somadores, outros codificadores, e outros ainda decodificadores. Contudo, o foco deles é resolver problemas maiores, como o acionamento ou não de um alarme de incêndio, um aviso de abertura de uma sequência de portas ou a contagem da quantidade de pessoas em uma loja. Caso surja alguma dúvida, não hesite em pesquisar no conteúdo já trabalhado nas seções anteriores.

Refita

Com o avanço da tecnologia digital, impulsionado pela miniaturização dos dispositivos e pela necessidade de diminuição do consumo de energia, tem havido uma busca por níveis de tensão cada vez mais baixos. Então, ao contrário dos 5 V utilizados anteriormente, agora se buscam tensões de 3,3 V, 1,8 V e até mesmo 1,2 V. Portanto, como novas tecnologias trazem sempre novos desafios, surgem algumas dúvidas: “é possível ligar uma saída de uma porta lógica de 3V3 em uma entrada 5 V?”; “há problema em ligar um circuito 3V3 TTL em um circuito 3V3 CMOS?”; “como fazer essa ligação?”.

Bem, como existe um pouco mais além das tecnologias TTL e CMOS, como o LTTL (*low power TTL*), BiCMOS, LVC (*low voltage CMOS*) etc., no entanto, para nosso estudo, basta saber que valores de *alimentação diferentes* vão alterar os valores de *limites de entrada e saída*. A saída para essa questão é sempre pesquisar os manuais (*datasheet*) e catálogos dos fabricantes, quando se pretende conduzir um projeto. A Figura 1.24 apresenta uma referência para os níveis lógicos de tensão entre algumas tecnologias:

Figura 1.24 | Famílias de CI e seus respectivos níveis de tensão



Fonte: Sacco (2020, [s.p.]).

Circuitos lógicos

Vamos conhecer, agora, as três portas lógicas mais utilizadas em circuitos digitais.

- *Porta OR (ou porta OU)*

Esta porta corresponde a um circuito que tem duas ou mais entradas e cuja saída é igual ao resultado da combinação destas. A Figura 1.25 mostra o símbolo lógico para uma porta OR. As entradas A e B são níveis lógicos de tensão, e a saída x é um nível lógico de tensão cujo valor é o resultado da operação OR entre A e B, ou seja, $x = A + B$

A expressão $x = A + B$ deve ser interpretada como: x é igual a A OU B, o que significa que x será 1 quando A ou B forem 1. Veja porta e tabela-verdade a seguir:

Figura 1.25 | Porta lógica OR (ou OU)

PORTA OR		
A	B	$X = A+B$
0	0	0
0	1	1
1	0	1
1	1	1

SÍMBOLO DE UMA PORTA OR

Fonte: adaptada de Tocci, Widmer e Moss (2003).

- *Porta AND (ou porta E)*

A saída da porta AND corresponde ao produto lógico AND das entradas, sendo $x = A \bullet B$, expressão que é lida como: x é igual a A E B, o que significa que x será 1 somente quando A e B forem, ambas, nível 1. O sinal (\bullet) é muitas vezes omitido e a expressão torna-se simplesmente $x = AB$.

Figura 1.26 | Porta Lógica AND (ou E)

PORTA AND		
A	B	$X = A \bullet B$
0	0	0
0	1	0
1	0	0
1	1	1

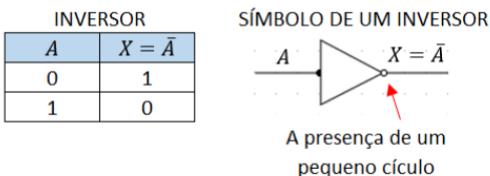
SÍMBOLO DE UMA PORTA AND

Fonte: adaptada de Tocci, Widmer e Moss (2003).

- Porta NOT ou Inversor

A operação NOT (ou inversor), ao contrário das operações OR e AND, pode ser realizada sobre uma única variável de entrada. Assim, se a variável A for submetida à operação de inversão, o resultado x pode ser descrito como $X = \bar{A}$ (A com uma barra em cima) – nesse caso, a barra sobre o nome da variável identifica **operação de inversão**. Essa expressão é lida como x é igual a A negado ou x é igual ao inverso de A, ou simplesmente A barrado.

Figura 1.27 | Porta lógica inversora (NOT)



Fonte: adaptada de Tocci, Widmer e Moss (2003).

A Figura 1.27 mostra o símbolo para o circuito NOT. Esse circuito sempre tem apenas uma entrada e seu nível lógico de saída é sempre o oposto ao nível lógico de entrada, a Figura 1.27 mostra ainda como um *inversor* afeta um sinal de entrada, invertendo sempre sua saída, de maneira que, se $A = 0$, sua saída é igual a 1, e vice-versa.

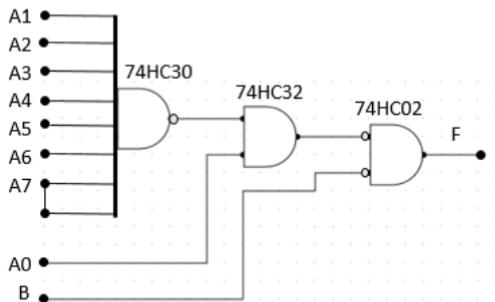
Assimile

Circuitos com portas AND, OR e NOT

Saiba que qualquer circuito lógico, independentemente de sua complexidade, pode ser construído pela combinação das três portas lógicas básicas: AND, OR e NOT (ou inversora). Antes de entendermos como projetar um circuito, é importante defini-lo. Segundo Vahid (2008), um circuito consiste em uma interconexão de componentes elétricos. Ou seja, trata-se aqui de diversos componentes conectados entre si. Vahid (2008, p. 28) deixa ainda mais claro o que é um circuito lógico: “Projetar um circuito que, para cada combinação diferente de entradas a e b, gere o valor apropriado de f”.

A Figura 1.28 é um exemplo de circuito lógico que utiliza um arranjo de portas lógicas de duas entradas A (entre 0 e 7) e B e uma saída F. No meio, existem três portas lógicas (74HC30, 74HC32 e 74HC02) que formam o circuito digital.

Figura 1.28 | Arranjo de portas lógicas



Fonte: adaptada de Pereira e Teixeira (2018).

Tabela-verdade

A proposta de uma tabela-verdade, como você viu, é figurar como uma técnica para ilustrar ou determinar de que forma a saída de um circuito vai depender dos níveis lógicos presentes nas suas entradas.

A Figura 1.29 relaciona, em tabela-verdade, todas as combinações possíveis para os níveis lógicos presentes nas entradas A e B com o correspondente nível lógico na respectiva saída x.

Figura 1.29 | Interpretação de uma tabela-verdade

PORTA	SÍMBOLO GRÁFICO	TABELA-VERDADE	FUNÇÃO LÓGICA	EXPRESSÃO															
E AND		<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>S</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	S	0	0	0	0	1	0	1	0	0	1	1	1	Assume valor 1 quando todas as entradas forem 1 e 0 nos outros casos.	$S = A \cdot B$
A	B	S																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
OU OR		<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>S</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	S	0	0	0	0	1	1	1	0	1	1	1	1	Assume valor 0 quando todas as entradas forem 0 e 1 nos outros casos.	$S = A + B$
A	B	S																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
NÃO NOT INVERSORA		<table border="1"> <thead> <tr> <th>A</th><th>S</th></tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>	A	S	0	1	1	0	Inverte o valor da variável de entrada A	$S = \bar{A}$									
A	S																		
0	1																		
1	0																		

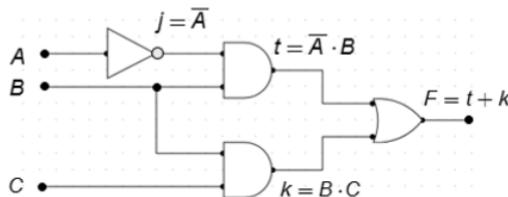
Fonte: adaptada de Pereira e Teixeira (2018).

Segundo Tocci, Widemer e Moss (2013), a melhor forma de entender o funcionamento de um circuito lógico é por meio de uma tabela-verdade, uma vez que ela permite:

- Analisar uma porta ou a combinação de cada conjunto lógica de cada vez.
- Encontrar um erro de forma mais fácil.

A Figura 1.30 descreve um circuito lógico completo. Analise em conjunto com a respectiva tabela-verdade e observe os valores intermediários entre as portas lógicas, pois eles são uma boa pista para solucionar a lógica final.

Figura 1.30 | Circuito lógico e respectiva tabela-verdade



A	B	C	$J = \bar{A}$	$t = \bar{A} \cdot B$	$k = B \cdot C$	$F = t + k$
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	1	1	0	1
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	1	1

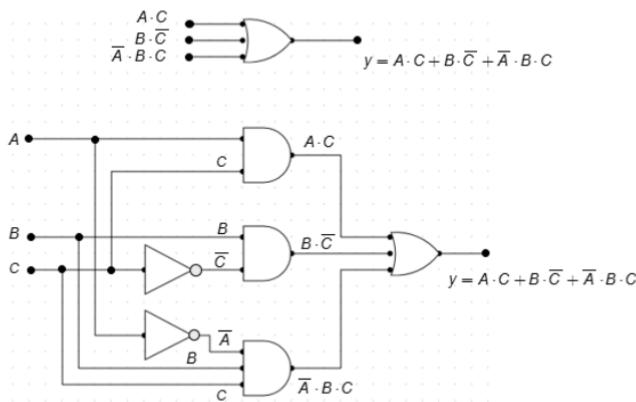
Fonte: adaptada de Pereira e Teixeira (2018).

Implementação de circuitos lógicos por expressão booleana

Quando a operação de um circuito é definida por uma expressão booleana, é possível, com base nela, determinar o respectivo diagrama de circuito lógico. Assim, se precisamos de um circuito definido por $X = A \bullet B \bullet C$, podemos presumir que será necessária uma porta AND de três entradas.

Se precisarmos de um circuito definido por $X = A + B$, poderemos usar uma porta OR de duas entradas com um *inversor* em uma delas. O mesmo raciocínio aplicado a esses casos simples pode ser estendido para circuitos mais complexos, conforme é ilustrado pela Figura 1.31.

Figura 1.31 | Arranjos de portas lógicas equivalentes



Fonte: adaptada de Tocci, Widmer e Moss (2003).

Podemos tomar como exemplo um circuito cuja saída seja $y = AC + BC + \bar{ABC}$. Essa expressão booleana contém três termos (AC, BC, \bar{ABC}) sobre os quais é aplicada a operação OR. Essa expressão nos diz que é necessária uma porta OR de três entradas, que são iguais a (AC, BC, \bar{ABC}). Isso está ilustrado na Figura 1.31, que traz uma porta OR de três entradas nomeadas (AC, BC, \bar{ABC}).

Cada entrada da porta OR tem um termo que é um produto lógico AND, o que significa que uma porta AND, com as entradas apropriadas, pode ser usada para gerar cada um desses termos. Isso é indicado na Figura 1.31, na parte em que mostra o diagrama final do circuito. Observe que o uso de inversores produz os termos \bar{A} e \bar{C} presentes na expressão.

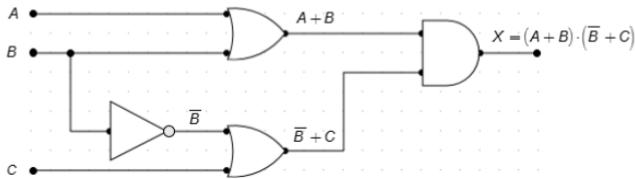
Exemplificando

Com base na expressão $X = (A + B)(\bar{B} + C)$, desenhe o diagrama do circuito lógico que a represente.

Solução:

Essa expressão mostra que os termos $A + B$ e $\bar{B} + C$ são entradas de uma porta END, e cada um deles é gerado pela porta OR independente. O resultado é demonstrado na Figura 1.32:

Figura 1.32 | Resultado gráfico da expressão $X = (A + B)(\bar{B} + C)$

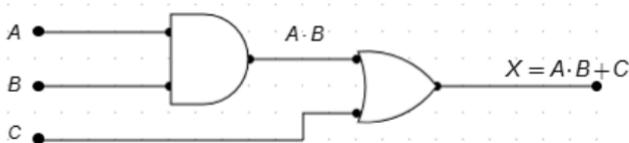


Fonte: adaptada de Tocci, Widmer e Moss (2003).

Obtenção da expressão booleana por um circuito lógico

Todos os circuitos lógicos podem ser descritos pelas operações booleanas básicas, independentemente da complexidade, porque as portas OR, AND e NOT são os blocos fundamentais dos sistemas digitais. Considerando a Figura 1.33 de um circuito com três entradas (A, B e C) e uma saída x, usando as expressões booleanas de cada porta, podemos determinar a expressão lógica da sua saída.

Figura 1.33 | Soma de uma variável com um produto

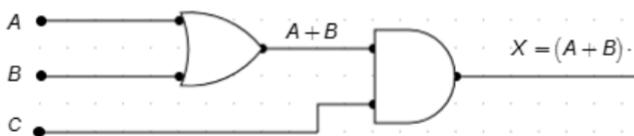


Fonte: adaptada de Tocci, Widmer e Moss (2003).

A expressão para a saída da porta AND escrita é $A \bullet B$, que está conectada a uma entrada da porta OR, que tem como outra entrada a variável C. A porta opera sobre as entradas de modo que sua saída é uma soma lógica destas. Dessa forma, podemos expressar a saída da porta OR como: $x = (A \bullet B) + C$, portanto: $x = A \bullet B + C$

De modo semelhante, podemos obter um novo arranjo ao trocar as portas lógicas de posição, embora o resultado obtido seja diferente. A expressão para a saída de uma porta OR é $x = A + B$, que está conectada em uma entrada da porta AND que tem como outra entrada a variável C. A porta opera sobre as entradas de modo que a saída é um produto lógico destas. Dessa forma podemos expressar a saída da porta AND como: $x = (A + B) \bullet C$, portanto: $x = A + B \bullet C$

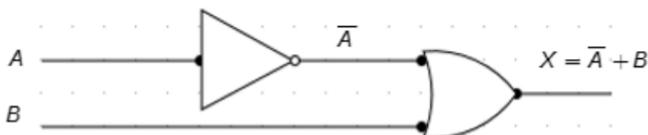
Figura 1.34 | Produto de uma variável com uma soma



Fonte: adaptada de Tocci, Widmer e Moss (2003).

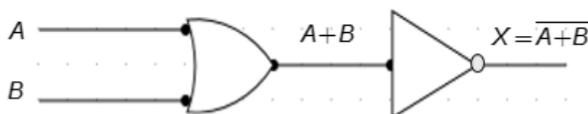
Por sua vez, quando encontrarmos um *inversor* em um circuito lógico, a expressão para a saída daquele será igual à expressão de entrada, com uma barra sobre ela. Assim, a entrada A é alimentada por meio de um *inversor*, cuja saída é, portanto, \bar{A} . A saída do *inversor* alimenta uma porta OR juntamente com B, de modo que a saída da OR é igual a $\bar{A} + B$. Observe que a barra está apenas sobre a variável A, significando que primeiro se inverte A e, em seguida, faz-se a operação OR com B, conforme pode ser visto na Figura 1.35.

Figura 1.35 | Porta lógica com inversor na entrada



Fonte: adaptada de Tocci, Widmer e Moss (2003).

Figura 1.36 | Porta lógica com inversor na saída



Fonte: adaptada de Tocci, Widmer e Moss (2003).

Por sua vez, na Figura 1.36, podemos ver a saída da porta OR igual a $A + B$, que é a entrada de um INVERSOR. Portanto, a saída do INVERSOR (que é a saída final) será igual a $(\bar{A} + \bar{B})$, visto que ele inverte a expressão inteira na sua entrada [observe que a barra cobre a expressão de entrada ($A + B$) inteira]. Isso é importante destacar porque, conforme veremos depois, as expressões $(\bar{A} + B)$ e $(\bar{A} + \bar{B})$ não são equivalentes. A expressão $(\bar{A} + B)$ significa que é realizada a operação OR entre A e B e, em seguida, a soma lógica é invertida, ao passo que a expressão $(\bar{A} + \bar{B})$ indica que A é invertida, B é invertida e o resultado é a operação OR dessas variáveis invertidas.

Saiba mais

Função NOT com (AND) = NAND

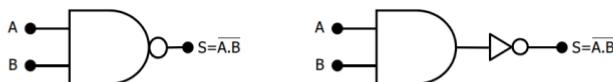
Na composição da função E com a função NÃO, a saída da função E é invertida: $S = \overline{A + B}$

Figura 1.37 | Tabela-verdade para porta NAND

A	B	$S = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

Fonte: adaptada de Tocci, Widmer e Moss (2003).

Figura 1.38 | Representação gráfica para porta NAND



Fonte: adaptada de Tocci, Widmer e Moss (2003).

Função NOT com (OR) = NOR

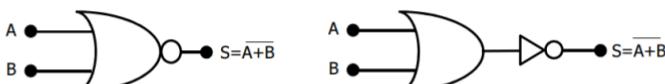
Na composição da função OU com a função NÃO, a saída da função OU é invertida: $S = \overline{A + B}$

Figura 1.39 | Tabela-verdade para porta NOR

A	B	$S = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

Fonte: adaptada de Tocci, Widmer e Moss (2003).

Figura 1.40 | Representação gráfica para porta NOR



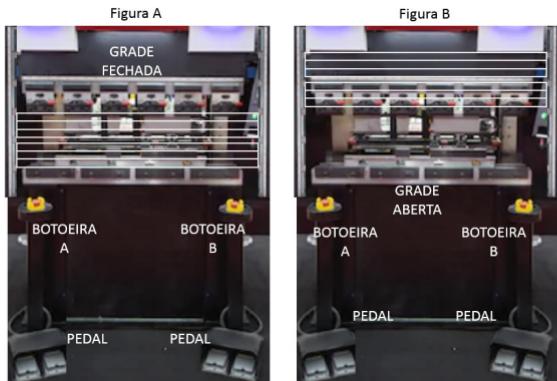
Fonte: adaptada de Tocci, Widmer e Moss (2003).

Sem medo de errar

Resolução da situação-problema 1

Para a resolução deste problema, vamos nos basear nos dados da Figura 1.41.

Figura 1.41 | Prensa



Fonte: adaptada de Shutterstock.

Para garantir a segurança do operador, é importante que ambas as botoeiras (botões A e B) sejam posicionados distantes uma do outra, a fim de garantir que o operador estará com as duas mãos ocupadas na tarefa. Além disso, estas outras premissas são necessárias:

- O sistema é ativado com nível lógico “1” e desativado com nível lógico “0”.
- O sistema ainda deve apresentar uma chave de “ligado ou desligado”, não considerada neste projeto.
- A grade de segurança deve ser a primeira variável a ser checada e ativada apenas na posição “grade fechada” (Figura-A) e desativada na posição “grade aberta” (Figura-B), embora logicamente a prensa só será acionada se todos as entradas estiverem ativas.
- Embora a figura apresente dois pedais de cada lado, o seu acionamento (com o pé) é único e o mesmo, não havendo a necessidade de usar os dois pés para acioná-lo. O duplo comando serve apenas como um conforto extra ao operador para (eventualmente) trocar de posição.
- Ambas as botoeiras precisam ser acionadas simultaneamente.
- Todas as entradas necessitam estar ativas (1) para a prensa funcionar, conforme tabela-verdade.

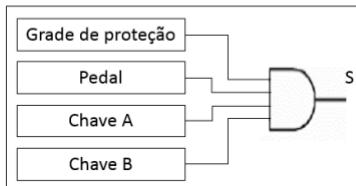
A seguir mostramos a tabela-verdade para acionamento da prensa:

GRADE	PEDAL	BOTÃO A	BOTÃO B	SAÍDA
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	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

A equação booleana, portanto, seria:

$$S = A \cdot B \cdot C \cdot D$$

O circuito equivalente seria uma porta AND com 4 entradas:



Resolução da situação-problema 2

Primeiramente vamos analisar cada situação proposta individualmente, considerando as entradas e saídas no circuito lógico e, assim, definir a base para criarmos a tabela-verdade.

Entradas:

Sensor de presença na rua A, registrando carro em A A = 1; logo, não havendo carros em A, $\rightarrow A = 0$.

Sensor de presença na rua B, registrando carro em B B = 1; logo, não havendo carros em B, $\rightarrow B = 0$.

Saídas:

Verde do semáforo 1 aceso $\rightarrow V1 = 1$

Verde do semáforo 2 aceso $\rightarrow V2 = 1$

Vermelho do semáforo 1 aceso $\rightarrow VM1 = 1$

Vermelho do semáforo 2 aceso $\rightarrow VM2 = 1$

Quando $V1 = 1$ temos:

$VM1 = 0 \text{ E } VM2 = 1 \text{ E } V2 = 0$

Quando $V2 = 1$ temos:

$VM2 = 0 \text{ E } VM1 = 1 \text{ E } V1 = 0$

Montando a tabela-verdade de entradas e saídas:

	A	B	V1	VM1	V2	VM2
Situação 1	0	0	0	1	1	0
Situação 2	0	1	0	1	1	0
Situação 3	1	0	1	0	0	1
Situação 4	1	1	1	0	0	1

Na situação 1, não temos carro em ambas as ruas. Nesse caso, vamos considerar que $V2$ (verde da rua B), por exemplo, permaneça aceso, isto é, que $V2=1$ – consequentemente: $VM2 = 0$, $VM1 = 1$, $V1 = 0$.

Na situação 2, temos carro apenas em B, logo $V2 = 1$ – consequentemente: $VM2 = 0$, $VM1 = 1$, $V1 = 0$.

Na situação 3, temos carro apenas em A, logo $V1 = 1$ – consequentemente: $VM1 = 0$, $VM2 = 1$, $V2 = 0$.

Na situação 4, temos carro em ambas as ruas, mas como A é rua preferencial, teremos a

mesma situação 3, logo $V1 = 1$ – consequentemente: $VM1 = 0$, $VM2 = 1$, $V2 = 0$.

Faça valer a pena

1. Existem várias formas e modelos para montar uma tabela-verdade. No entanto, a correta elaboração dela poderá ajudar no entendimento e na simplificação no momento de elaborar qualquer projeto de lógica booleana.

Com base nas premissas de uma tabela-verdade, assinale a alternativa que representa a tabela da expressão: $S = A + B + C$

a.

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

b.

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

c.

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

d.

A	B	C	S
0	0	0	1
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

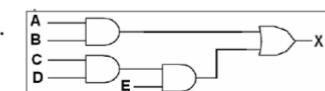
e.

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

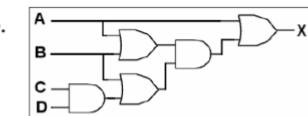
2. Todos os computadores modernos atualmente se utilizam da álgebra booleana, representada pelas portas lógicas inseridas no interior dos circuitos integrados. Essas portas, quando combinadas, formam arranjos com a lógica desejada para operar qualquer circuito digital.

Com base na álgebra booleana, assinale a seguir a alternativa que apresenta corretamente o arranjo das portas lógicas dado pela expressão $X = A \bullet B + (C \bullet D \bullet E)$

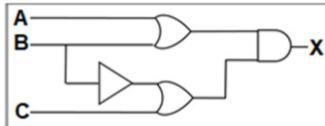
a.



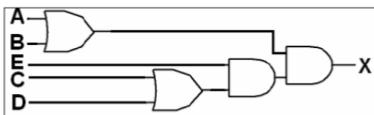
b.



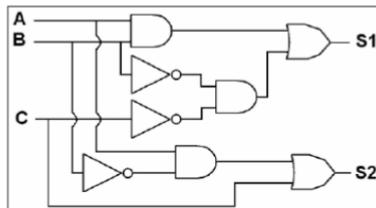
c.



d.



e.



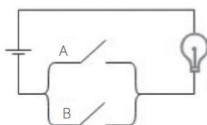
3. As três principais portas lógicas utilizadas em circuitos digitais – porta AND (lógica E), OR (lógica OU) e NOT (lógica NÃO ou INVERSORA) são apresentadas a seguir:

PORTA	SÍMBOLO GRÁFICO	TABELA-VERDADE	FUNÇÃO LÓGICA	EXPRESSÃO															
E AND		<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>S</th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	S	0	0	0	0	1	0	1	0	0	1	1	1	Assume valor 1 quando todas as entradas forem 1 e 0 nos outros casos.	$S = A \cdot B$
A	B	S																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
OU OR		<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>S</th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	S	0	0	0	0	1	1	1	0	1	1	1	1	Assume valor 0 quando todas as entradas forem 0 e 1 nos outros casos.	$S = A + B$
A	B	S																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
NÃO NOT INVERSORA		<table border="1"> <thead> <tr> <th>A</th><th>S</th></tr> </thead> <tbody> <tr> <td>0</td><td>1</td></tr> <tr> <td>1</td><td>0</td></tr> </tbody> </table>	A	S	0	1	1	0	Inverte o valor da variável de entrada A	$S = \bar{A}$									
A	S																		
0	1																		
1	0																		

A presença de um pequeno círculo sempre indica ser uma inversora

Analise a figura seguinte, considere o texto-base anterior e, com base nisso, escolha a alternativa correta:

(a)



(b)



- a. Em (a) há uma porta lógica OR. A lâmpada ficará acesa se e somente se ambas as entradas forem acionadas. Em (b) há uma porta lógica AND. A lâmpada ficará acesa eventualmente quando ambas as entradas forem acionadas.
- b. Em (a) há uma porta lógica AND. A lâmpada ficará acesa se ambas as entradas forem acionadas. Em (b) há uma porta lógica OR. A lâmpada ficará acesa se A ou B ou ambas as entradas forem acionadas.
- c. Em (a) há uma porta lógica OR. A lâmpada ficará acesa se A ou B ou ambas forem acionadas. Em (b) há uma porta lógica AND. A lâmpada ficará acesa se e somente se ambas as entradas forem acionadas.
- d. Tanto (a) quanto (b) podem ligar a lâmpada independentemente da posição na qual estão as portas A e B.
- e. Não é possível abstrair portas lógicas observando o desenho. O que se conclui é que independentemente da posição das chaves as lâmpadas não serão acesas.

Referências

AULA III - SIMPLIFICAÇÃO EXPRESSÕES BOOLEANAS. [S.l.; s.n], 2017. 1 vídeo (28:40 min). Publicado pelo canal Nivaldo Junior. Disponível em: <https://www.youtube.com/watch?v=4nVoLN7cBAw>. Acesso em: 1 jun. 2020.

BOLLMANN, A. **Fundamentos da Automação Industrial Pneutrônica**. Projetos de comandos binários eletropneumáticos. ABHP, São Paulo, 1997.

FLOYD, T. **Sistemas digitais**: fundamentos e aplicações. 9. ed. Porto Alegre: Bookman, 2007.

FOROUZAN, B. A. **Comunicação de dados e redes de computadores**. 4. ed. Porto Alegre: AMGH, 2008.

GIVANT, S.; HALMOS, P. **Introduction to Boolean Algebras**. Undergraduate Texts in Mathematics. New York: Springer, 2009.

IDOETA, I. V.; CAPUANO, F. G. **Elementos de eletrônica digital**. 41. ed. São Paulo: Érica, 1997.

NASCIMENTO, J. N. R. do. **Microcontroladores e microprocessadores**. Londrina: Editora e Distribuidora Educacional S.A, 2017.

PEDRONI, V. A. **Eletrônica digital moderna e VHDL**. São Paulo: Elsevier, 2010.

PEREIRA, R. V. M.; ROCHA FILHO, E. **Sistemas digitais avançados**. Londrina: Editora e Distribuidora Educacional S.A, 2017. Disponível em: <https://biblioteca-virtual.com/detalhes/livro/1512>. Acesso em: 4 jun. 2020.

PEREIRA, R. V. M.; TEIXEIRA, H. T. **Circuitos digitais**. Londrina: Editora e Distribuidora Educacional S.A, 2018. Disponível em: <https://biblioteca-virtual.com/detalhes/livro/512>. Acesso em: 4 jun. 2020.

SACCO, F. Níveis e limites de tensões digitais. **Embarcados**, 2020. Disponível em: <https://bit.ly/2XKakA4>. Acesso em: 24 jun. 2020.

TEIXEIRA, H. T.; TAVARES M. F.; PEREIRA, R. V. M. **Sistemas digitais**. Londrina: Editora e Distribuidora Educacional S.A, 2017. Disponível em: <https://biblioteca-virtual.com/detalhes/livro/1571>. Acesso em: 4 jun. 2020.

TOCCI, R. J.; WIDMER S. N.; MOSS, L. G. **Sistemas Digitais**: princípios e aplicações. 8. ed. São Paulo: Pearson Education/Prentice Hall, 2003.

TOCCI, R. J.; WIDMER S. N.; MOSS, L. G. **Sistemas digitais**: princípios e aplicações. 11. ed. São Paulo: Pearson Education/Prentice Hall, 2011.

VAHID, F. **Sistemas Digitais**: projeto, otimização e HDLS. Porto Alegre: Bookman, 2008.

Unidade 2

Andrea Acunha Martin

Circuitos lógicos combinacional e sequencial

Convite ao estudo

Olá, aluno! Seja muito bem-vindo!

Você já parou para pensar que, neste exato momento em que você está estudando, milhões de pessoas estão vendo um vídeo na internet, usando um *smartwatch*, trabalhando no computador e acessando o celular? E você sabe o que tudo isso tem em comum? A eletrônica digital! Todos esses equipamentos, que certamente você usa ou já ouviu falar, só puderam ser criados devido à evolução dos circuitos digitais.

Anteriormente, você estudou sobre os circuitos digitais e a álgebra booleana e viu os conceitos fundamentais sobre sistemas lógicos. Para darmos seguimento em seu aprendizado, inicialmente serão abordadas as características dos sistemas analógicos e dos digitais e as diferenças entre eles. Também conceituaremos a lógica combinacional e a sequencial, apontando a principal diferença entre elas e o quanto são essenciais para os sistemas eletrônicos digitais atuais.

Em seguida, estudaremos os circuitos combinacionais de forma mais profunda, assim veremos que as portas lógicas básicas podem ser combinadas para realizar atividades importantes para o funcionamento dos computadores, como os circuitos somadores e os circuitos subtratores. Também aprenderemos sobre circuitos comparadores, codificadores e decodificadores, utilizados quando é necessário realizar conversões de um tipo de código para outro, e sobre circuitos multiplexadores e demultiplexadores, presentes em telecomunicações.

Por fim, veremos os fundamentos da lógica sequencial, os dispositivos lógicos biestáveis e o *flip-flop*, unidade básica para construção de contadores, registradores e que pode ser usado na construção de algumas memórias.

Hoje você saberia dizer onde encontramos os circuitos lógicos e os combinacionais? Se não, não se preocupe, logo você descobrirá. Ao final desta seção, você será capaz de compreender os principais conceitos de lógica combinacional e de lógica sequencial, assim como projetar esses tipos de

circuitos. Espero que você goste deste assunto e o ache bastante interessante e útil para sua vida. Estão preparados? Bons estudos!

Seção 1

Introdução à lógica combinacional e sequencial

Diálogo aberto

Os sistemas digitais estão por toda parte: em nosso trabalho, conosco e até em nossas casas. Um exemplo disso são os computadores. Nós sabemos da importância de um computador, mas como será que ele funciona? O computador é um dispositivo eletrônico digital, ou seja, ele é capaz de armazenar, processar e gerar dados na forma digital. Porém, o computador não consegue processar dados analógicos, e a solução é a conversão deles para o formato digital. Mas o que é um dado digital? E um dado analógico? Nesta seção você verá as diferenças entre sistemas analógicos e digitais, bem como alguns exemplos de cada um desses sistemas.

Também sabemos que os computadores realizam operações lógicas e aritméticas, mas quem são os responsáveis por isso? E por outras funções necessárias, como a multiplexação e a decodificação? Os elementos responsáveis são chamados de circuitos combinacionais, formados por várias portas lógicas e cuja saída depende diretamente dos valores atuais das entradas. Por exemplo, um multiplexador é um circuito combinacional e podemos encontrá-lo nos relógios digitais que mostram a hora, os minutos e/ou os dias e o mês. Aliás, nos relógios digitais também temos os decodificadores, os quais são também circuitos combinacionais. Os multiplexadores, se combinados com outros circuitos digitais, podem controlar um processo industrial através do sequenciamento de operações, por exemplo, e misturar três ingredientes. Uma aplicação prática dos demultiplexadores ocorre no sistema de monitoração e segurança.

Outro elemento fundamental nos computadores são os circuitos sequenciais, compostos por um circuito combinacional e por elementos de memória. Esses elementos de memórias são os *latches* e *flip-flops*. A principal característica dos circuitos sequenciais é a saída, a qual depende dos valores das entradas e dos estados dos *latches* ou *flip-flops* utilizados. Os *flip-flops* são geralmente encontrados como parte de equipamentos mais complexos. Um exemplo disso pode ser observado no próprio computador. Qualquer entrada de teclado ou entrada de *bluetooth* tem um *buffer*, o qual é um registrador de deslocamento, que é um circuito sequencial. Está achando tudo isso muito complicado? Não se preocupe, pois veremos em detalhes o que é lógica combinacional e sequencial, quais são as principais diferenças entre elas e algumas aplicações.

A eletrônica digital está presente nos computadores, em sistemas que usam microcontroladores, em aparelhos eletroeletrônicos, em celulares e até nos brinquedos. Em 1980, foi lançado o Genius, que foi anunciado como sendo o computador que brinca. Em 1983, o robô Arthur tinha falas prontas e andava com comandos de um controle remoto. As crianças adoravam!

Saiba mais

O Genius, que, nos Estados Unidos da América, era chamado de Simon, é um jogo que fez muito sucesso na década de 1980. Esse jogo foi criado por um engenheiro eletrônico e consultor de engenharia Ralph H. Baer. Em 1975, Baer começou um negócio de consultoria independente e começou a trabalhar junto com Marvin Glass & Associates, empresa de design de brinquedos. Baer desenvolvia brinquedos e jogos eletrônicos. Marvin e Baer começaram, então, a criar um jogo móvel que possuía quatro botões coloridos, sendo que cada botão tocava uma nota diferente. O objetivo do jogo era repetir uma sequência cada vez mais longa de tons que era inventada. Se o jogador repetisse a ordem errada, ele perdia. Os sons que o jogo emitia são os mesmos sons que uma corneta pode tocar, quatro notas. No Brasil, o jogo foi chamado de Genius e foi relançado em 2012.

O jogo Genius pode ser construído através de um microcontrolador, como o Arduino, que possui hardware e software livres.

Considere agora que você é o mais novo engenheiro contratado por uma grande fábrica de brinquedos para ser o responsável por toda parte da eletrônica digital dos brinquedos.

Um novo produto para entreter as crianças foi lançado: o robô *Pet Dog*, um brinquedo que anda, tem sensores infravermelhos, late e tira fotos! Ele responde a comandos de voz e ao controle remoto. Porém, após lançado, o produto apresentou falhas e você foi alocado para resolvê-las.

Uma das características do *Pet Dog* é latir quando dois dos três sensores presentes no focinho e em cada lateral do corpo é ativada. Isso ocorre toda vez que passa alguma coisa por eles e uma das falhas está justamente no fato de o robô não estar latindo.

O seu primeiro trabalho é projetar novamente o circuito que faz com que o robô *Pet Dog* lata toda vez que dois dos seus três sensores estiverem ativados.

Como uma melhoria nesse projeto, considere também que o *Pet Dog* tem um microfone embutido para os comandos de voz com um circuito conversor A/D acoplado. Esse circuito tem dois pinos lógicos extras de saída

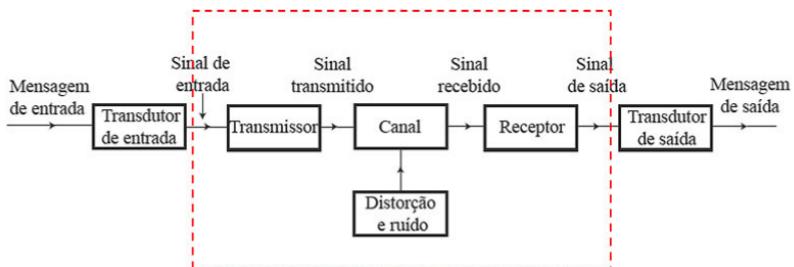
para indicar apenas o nível de ruído do ambiente e, dessa forma, apesar do processamento da voz, que é mais complexo, há a possibilidade de haver dois níveis indicadores de ruídos do ambiente. Assim, o *Pet Dog* deve latir ou não levando em consideração dois níveis de ruído: latir muito para um ruído altíssimo e não latir para a ausência de som ambiente, independentemente dos outros sensores. Dessa forma, além de um comportamento mais robusto, o ato de latir se torna mais direto pois depende de sensores lógicos binários.

Diariamente surgem novas tecnologias, que se tornam parte da nossa vida. Diante disso, os sistemas digitais têm uma enorme importância no panorama tecnológico atual e essa é a maior motivação para que você se dedique e busque cada vez mais informações sobre os Sistemas Digitais e Microprocessadores. Bons estudos!

Não pode faltar

A expansão das tecnologias que envolvem comunicação digital foi extremamente rápida. A internet, que antes era uma palavra familiar apenas a técnicos e à comunidade científica, hoje é conhecida por todos. As comunicações, que antes eram feitas por pombos-correios, luzes e fogo, foram substituídos por sinais elétricos capazes de transmitir a longas distâncias e à velocidade da luz. Um modelo típico de um sistema de comunicação possui transmissor, canal, receptor e destino (Figura 2.1).

Figura 2.1 | Sistema de comunicação



Fonte: adaptada de Lathi e Ding (2012).

Os principais componentes de um sistema de comunicação são, segundo Lathi e Ding (2012):

- **Transmissor** ou emissor modifica o sinal em banda base para que se tenha uma transmissão eficiente. Alguns exemplos de transmissores são: conversor A/D, codificador e modulador.

- **Canal** ou meio é por onde os sinais elétricos na saída do transmissor são transportados até o receptor e que, de maneira geral, atenua o sinal e distorce a forma de onda. Exemplos de canal: par de fios de cobre trançados, cabo coaxial, fibra óptica e enlace de rádio.
- **Receptor** processa o sinal recebido do canal, revertendo as modificações feitas pelo transmissor e removendo as distorções causadas pelo canal. O receptor pode ser um demodulador, um decodificador ou um conversor D/A.
- **Destino** para onde a mensagem é endereçada.

Os sinais que passam por canais de comunicação, além de sofrerem distorções do canal, também apresentam ruídos, ou seja, interferências e perturbações indesejáveis que são aleatórias e imprevisíveis. Esses sistemas de comunicação podem ser classificados em analógicos e em digitais, porém, antes de adentramos esse âmbito, é importante compreendermos as quantidades.

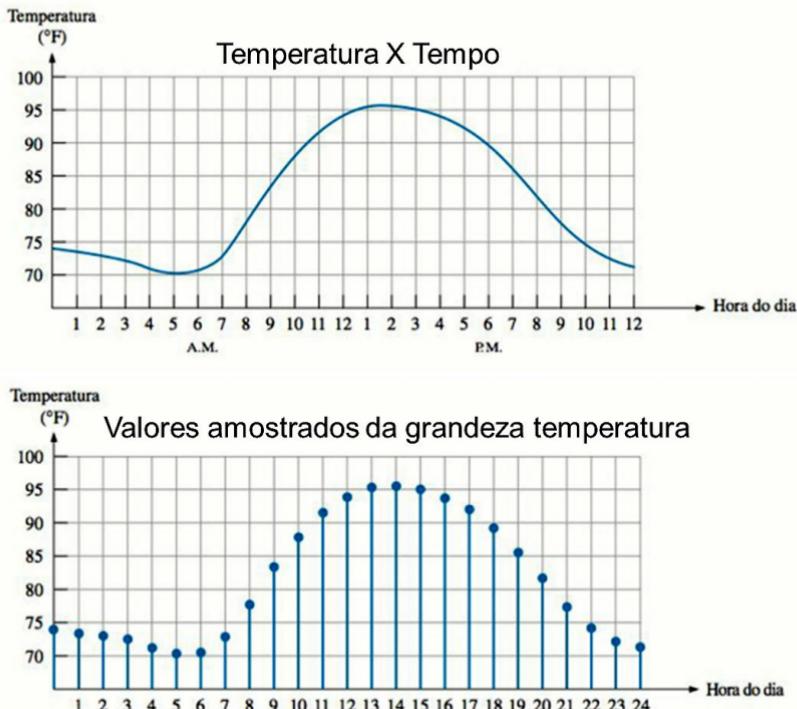
Assimile

Define-se sinal como um fenômeno físico que possui um único valor em cada instante de tempo.

Quantidades podem ser medidas, monitoradas, armazenadas, manipuladas, etc. Elas podem ser representadas de duas formas: quantidade analógica ou digital. A principal característica das quantidades analógicas é a de variar ao longo de uma faixa contínua de valores, enquanto as quantidades digitais são representadas por símbolos chamados dígitos, e o tempo varia em degraus. E onde isso se aplica? A velocidade de um automóvel, que, dependendo do modelo, pode variar entre 0 km/h até 200 km/h, é uma “quantidade analógica”. Já sobre quantidade digital, podemos citar o relógio digital. Vamos explorar mais essas quantidades?

Um exemplo de uma grandeza no formato analógico é a temperatura (Figura 2.2). Notamos que, no decorrer do dia, a temperatura passa por variações. Observe que, no segundo gráfico, é possível verificar os valores amostrados da grandeza analógica, a qual é apresentada no primeiro gráfico.

Figura 2.2 | Grandeza analógica: temperatura *versus* tempo e valores amostrados da grandeza temperatura



Fonte: Floyd (2007, p. 20-21).

Nota-se que cada valor representado por um ponto pode ser digitalizado e representado através de números binários: 1s e 0s. Por exemplo, se pegarmos o ponto que representa as 5 horas, o valor é 70 em decimal e digitalizado, ou seja, em binário 1000110.

Agora que já sabemos as formas como as quantidades são representadas, podemos definir sistema digital e sistema analógico.

Um sistema digital é um conjunto de dispositivos criados para manipular informações que são representadas no formato digital e que, portanto, usam valores discretos, ou seja, descontínuos. Já um sistema analógico utiliza um intervalo contínuo de valores e, portanto, compreende os dispositivos que manipulam quantidades físicas analógicas.

Assimile

Podemos exemplificar um sinal analógico através da temperatura física, pois é um valor contínuo e pode ter um valor medido de 30,231456... graus. Já o sinal digital, pode ser exemplificado pelo número de dedos, pois o valor obrigatoriamente só pode ser 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ou 10, ou seja, temos um conjunto finito de valores (VAHID, 2008).

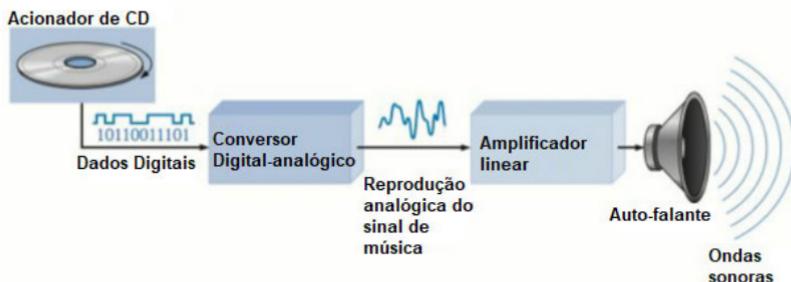
Dentre as vantagens de se utilizar os sistemas digitais, temos: a facilidade de se projetar, o armazenamento simples das informações, maior precisão e exatidão, a diminuição do ruído, a possibilidade de utilizar programação, entre outras. A grande limitação das técnicas digitais está no fato de que o mundo real é praticamente analógico, porém isso pode ser contornado convertendo as entradas analógicas para o formato digital. Após esse passo, a informação digital é processada e a saída é convertida para o formato analógico.

Mas existem limitações? Podemos dizer que qualquer fenômeno analógico pode ser digitalizado: por exemplo, os automóveis digitalizam informações sobre a velocidade do carro, o nível de combustível e outros. Isso só é possível pois se tem um computador montado em um único circuito integrado que é capaz de monitorar e controlar o veículo. Para ocorrer a digitalização de fenômenos analógicos, é necessário basicamente (VAHID, 2008):

- Um sensor que meça o fenômeno físico analógico e que converta o valor medido em um sinal elétrico.
- Um conversor analógico-digital que converta o sinal elétrico em códigos binários. O conversor deve amostrar (medir) o sinal elétrico a uma taxa regular e converter cada amostra em um valor de bits.
- Um conversor digital-analógico que converta bits de volta à forma de sinal elétrico.
- Um atuador que converta o sinal elétrico de volta à forma de fenômeno físico.
- Sensores e atuadores que juntos são tipos de dispositivos conhecidos como transdutores (dispositivos que convertem uma forma de energia em uma outra).

Há alguns aparelhos que utilizam tanto circuitos digitais quanto analógicos, como o aparelho de CD (Figura 2.3). A música, que está no formato digital, é armazenada no CD.

Figura 2.3 | Diagrama em bloco de um aparelho de CD



Fonte: Floyd (2007, p. 21).

Um sistema ótico, através do diodo laser, recebe os dados digitais a partir do disco girante e os transfere para um conversor digital-analógico, convertendo os dados digitais em um sinal analógico, que nada mais é do que uma reprodução elétrica da música original. Após a conversão, esse sinal é amplificado e enviado ao alto-falante. Já quando a música é gravada no CD, ocorre o processo inverso e é usado um conversor analógico-digital (FLOYD, 2007).

Agora que já estudamos sobre as diferenças entre sistemas analógicos e digitais, vamos apreender a lógica combinacional e sequencial.

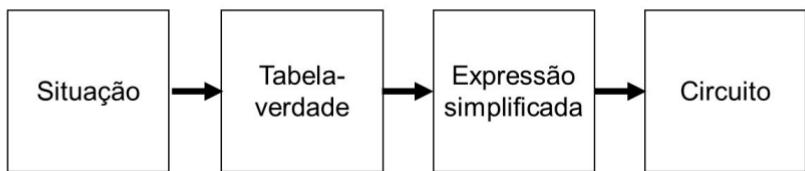
Podemos dividir os sistemas digitais em, basicamente, dois grupos: lógica combinacional e lógica sequencial. Quando as portas lógicas são interconectadas, elas podem produzir uma saída específica para as diferentes combinações possíveis das variáveis de entrada. Esse circuito resultante é chamado de lógica combinacional. O circuito lógico combinacional é utilizado para resolver problemas nos quais as saídas dependem exclusivamente da combinação dos valores da entrada.

Para construir um circuito combinacional, é necessário ter as expressões características dele, as quais são obtidas pelas tabelas-verdade. Os circuitos combinacionais são capazes de implementar sistemas mais simples. Para os sistemas com circuitos mais complexos, os combinacionais servem de base.

Alguns exemplos de circuitos combinacionais são os codificadores, os decodificadores, os circuitos aritméticos e outros derivados. Esse tipo de circuito é encontrado, na prática, nos circuitos integrados comerciais ou internos e nos circuitos integrados dedicados.

Para ajudar no projeto, há uma sequência para obtenção de um circuito combinacional (Figura 2.4).

Figura 2.4 | Sequência para obtenção de um projeto de circuito combinacional



Fonte: Capuano (2014, p. 67).

Dessa forma, fica evidenciado que, por meio da situação pedida, obtém-se a tabela-verdade e, a partir dela, aplicando as simplificações e demais técnicas, encontra-se a expressão simplificada, a qual se aplica no circuito final.

Exemplificando

Alguns vagões de trens de alta velocidade normalmente possuem, nas cabines de banheiro, um sinal luminoso que indica se estão desocupadas. Suponha que o vagão de um trem possua quatro banheiros e que cada um tenha um sensor que emite um sinal luminoso, indicando, quando aceso, que está ocupado e, quando apagado, que está desocupado.

Para resolver esse problema, vamos considerar o valor 1, sinal luminoso aceso, e 0, sinal luminoso apagado. Dessa forma, é possível montar a tabela-verdade (Tabela 2.1). Como temos quatro banheiros (A, B, C e D), temos quatro entradas. A saída S será 1 sempre que tivermos pelo menos um banheiro desocupado.

Tabela 2.1 | Tabela-verdade do problema dos banheiros no trem

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

1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Fonte: elaborada pela autora.

Após montarmos a tabela-verdade, observamos que a saída só será 0 quando todos os banheiros estiverem ocupados, sendo assim basta uma porta OR de quatro entradas para montar o circuito.

Nem sempre é possível resolver projetos em sistemas digitais utilizando circuitos combinacionais, pois algumas vezes pode ser necessário conhecer um ou mais estados anteriores. Sendo assim, os circuitos sequenciais se diferem dos circuitos combinacionais quando possuem saídas dependentes das variáveis de entrada e/ou de seus estados anteriores, que permanecem armazenados. Os estados anteriores podem realimentar as entradas e esse tipo de sistema é acionado sequencialmente por pulsos. Exemplos de dispositivos que utilizam lógica sequencial são os *flip-flops*, que, quando devidamente interligados, formam os contadores síncronos e assíncronos, os registradores de deslocamento e as memórias eletrônicas. Dessa forma, esse tipo de circuito é essencial na arquitetura interna dos sistemas computadorizados (CAPUANO, 2014).

Assimile

O circuito combinacional é aquele no qual a saída depende única e exclusivamente das possíveis combinações entre as variáveis de entrada.

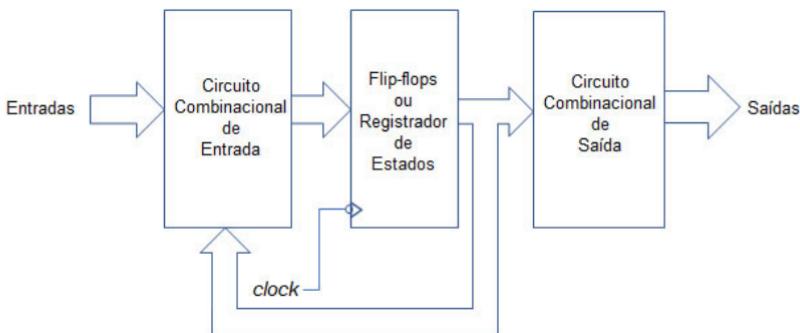
E, no circuito sequencial, a saída depende da entrada e/ou dos seus estados anteriores, ou seja, possui memória.

Um sistema digital pode ser implementado através de dois métodos: por meio da programação de um microprocessador ou por meio da criação de um próprio circuito digital, feito especialmente para atender a uma necessidade.

Os sistemas sequenciais, que são projetados para aplicações específicas, também podem ser conhecidos como máquinas de estado ou máquinas de estados finitos. Um exemplo são os contadores, que podem ser classificados como um caso particular de máquinas de estado. Há dois métodos para a implementação de máquinas de estados, os modelos de Moore e os de Mealy.

Quando as saídas dependem do estado atual registrado no sistema, temos o modelo de Moore, no qual as mudanças das saídas ocorrem sempre na próxima descida de *clock* (Figura 2.5). Podemos entender esses modelos como fluxogramas.

Figura 2.5 | Sistema utilizando o modelo de Moore

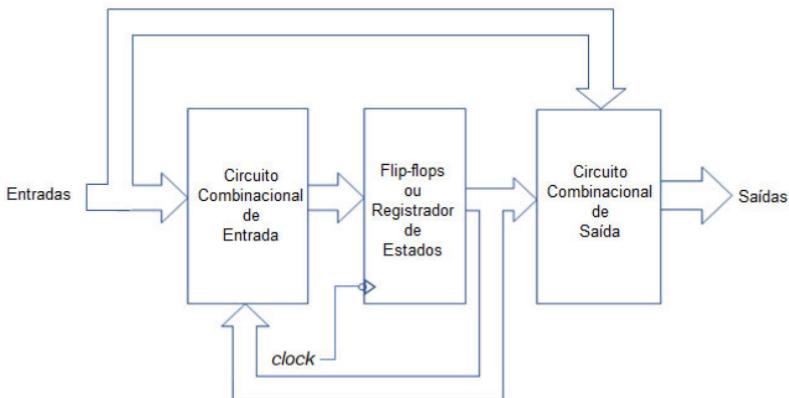


Fonte: adaptada de Capuano (2014).

Podemos observar que o *clock* controla o registrador que envia os estados, armazenando os valores das entradas e o estado anteriormente existente para o estágio de saída. Dessa forma, somente na próxima descida do *clock*, as saídas assumirão um novo estado.

No modelo de Mealy, as saídas dependem das entradas e do estado atual. Caso aconteça uma mudança nas entradas, as saídas também mudarão. A Figura 2.6 apresenta o diagrama de blocos de um sistema sequencial utilizando o modelo de Mealy (CAPUANO, 2014).

Figura 2.6 | Sistema utilizando o modelo de Mealy



Fonte: adaptada de Capuano (2014).

Notamos, na Figura 2.6, que o *clock* controla o registrador que envia os estados, armazenando o resultado das entradas e o estado anteriormente existente para o estágio de saída. Nesse modelo, as saídas também recebem as informações que vêm diretamente das entradas e que podem ser alteradas, mesmo antes da atuação do *clock*, pela mudança dos estados de entradas.

As máquinas de estado podem ser também classificadas em sistemas síncronos ou assíncronos.

Para realizar projetos de sistemas sequenciais devemos determinar o diagrama de estados, levantar a tabela-verdade, colocar nela os estados a serem assumidos pelas entradas utilizando a tabela de projetos, passar os casos de saída para os mapas de Karnaugh, para poder simplificá-los, e, finalmente, projetar o circuito com as expressões simplificadas (CAPUANO, 2014).

Refita

Podemos descrever um computador, basicamente, como um dispositivo que recebe informações de entrada, realiza um processamento e as entrega nas saídas. Podemos definir um computador como sendo um sistema digital sequencial programável?

Nesta seção vimos as características e as diferenças dos sistemas analógicos e dos sistemas digitais; a diferença da lógica combinacional e da sequencial e o quanto esses circuitos são essenciais para os sistemas eletrônicos digitais atuais. Ainda, vimos os sistemas sequenciais que são projetados para aplicações específicas, também conhecidos como máquinas de estado

ou máquinas de estados finitos, e vimos que existem dois métodos para a implementação de máquinas de estados, os modelos de Moore e os de Mealy.

A partir da próxima seção, todos esses conceitos ficarão mais claros quando forem apresentados, na prática, os tipos de circuitos e o seu funcionamento.

Bons estudos!

Sem medo de errar

Agora que você já adquiriu os conhecimentos necessários, vamos relembrar as falhas que o *Pet Dog* teve após o lançamento e as melhorias que você vai implantar para deixar o brinquedo mais atrativo? Uma das características do *Pet Dog* é latir quando dois dos três sensores presentes no focinho e em cada lateral do corpo são ativados, função que não está sendo realizada. Você também deverá incrementar o projeto para o *Pet Dog* latir de acordo com o ruído presente no ambiente (latir muito para altíssimo ruído e não latir para ausência de som ambiente, independentemente dos outros sensores).

Nesse sentido, para que o ajuste e a melhoria sejam realizados, você apresentará para o seu superior: a **tabela-verdade** com os sensores; a **expressão que executa** essa tabela; a **simplificação da equação** (utilize o mapa de Karnough) e o **circuito finalizado**.

Primeiramente, você monta a tabela-verdade, que nada mais é do que as saídas de um sistema para todas as combinações possíveis de entrada. Por meio desta, é possível definir se a proposição é verdadeira ou falsa. Mas, para isso, vamos definir quantas combinações teremos nessa tabela. Como temos três sensores instalados no *Pet Dog* e mais um que captará o nível de ruído, a tabela-verdade deverá ter quatro entradas. Para quatro entradas, temos: $2^4 = 16$ combinações.

A saída será definida em nível alto (1), quando o *Pet Dog* latir, e nível baixo (0), quando ele não latir. Dessa forma, montamos a tabela-verdade a seguir (Tabela 2.2):

Tabela 2.2 | Tabela-verdade latido *Pet Dog*

Sensor focinho (SF)	Sensor lado direito (SD)	Sensor lado esquerdo (SE)	Nível de ruído (NR)	Late (L)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0

0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

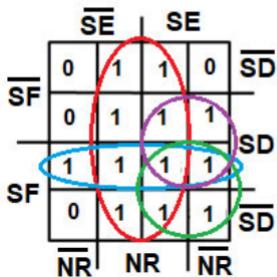
Fonte: elaborada pela autora.

O próximo passo é obter a expressão que executa a tabela-verdade, ou seja, em que as expressões e as saídas (*Pet Dog, late*) estejam em nível 1. Portanto temos:

$$L = \overline{SFSDSENR} + \overline{SFSDSEN}R + \overline{SFSD}SENR + \overline{SFSD}SEN\overline{R} + \overline{SFSD}SEN\overline{R} + SF\overline{SD}SENR + \\ SF\overline{SD}SEN\overline{R} + SF\overline{SD}SEN\overline{R} + SF\overline{SD}SEN\overline{R} + SF\overline{SD}SEN\overline{R} + SF\overline{SD}SEN\overline{R}$$

O terceiro passo é colocar, no mapa de Karnaugh, a expressão que executa a tabela-verdade (Figura 2.7). Como temos quatro entradas o mapa de Karnaugh também terá quatro entradas. Para cada expressão retirada da tabela-verdade colocamos o valor 1 e nas posições que sobrarem o valor 0.

Figura 2.7 | Mapa de Karnaugh do latido do *Pet Dog*



Fonte: elaborada pela autora.

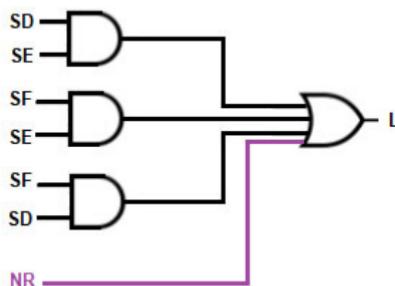
Para simplificar o mapa, já visto em outra unidade, devemos agrupar o maior número possível de uns, desde que eles estejam adjacentes a outros uns (2, 4, 8 ou 16 para um mapa de 4 entradas). Lembrando que todos os uns devem entrar na simplificação, mesmo que não seja possível agrupá-los. Esses termos são chamados de isolados. Diante disso, em vermelho, temos 8 uns, e o termo resultante é NR , pois é o único termo que se repete nos 8 números uns agrupados. A próxima simplificação possível é um quarteto que está em roxo. Lembre-se de que, para um melhor agrupamento, podemos repetir o número 1, que já foi agrupado. Para esse agrupamento, o termo resultante fica: $S = SESD$. O próximo quarteto é o que está em verde, e o resultado é: $SFSE$. Como ainda faltou um número 1, devemos agrupá-lo com os outros três, formando um novo quarteto em azul. O resultado é: $SFSD$. Com todos os termos que foram gerados, formamos um único através de ORs.

$$S = NR + SESD + SFSD + SFSE$$

O último passo é desenhar o circuito, Figura 2.8, através da expressão simplificada. Observando, temos três portas AND e o termo NR , os quais se juntam em uma única porta OR.

O circuito é dado por:

Figura 2.8 | Circuito latido do Pet Dog



Fonte: elaborada pela autora.

Dessa forma, foi possível identificar os sensores e você pôde ajustar as falhas para o relançamento do produto. Agora, você pode informar à empresa que deverá ser realizado o *recall* do circuito lógico corrigido e implementado por você.

Avançando na prática

Cofre de uma relojoaria

Uma relojoaria que funciona dentro de um shopping, com o expediente de 10h até 22h, possui dois gerentes GA e GB. Como a relojoaria estava tendo problemas de segurança, você foi chamado para resolvê-lo.

Para aumentar a segurança da loja, sua sugestão foi fazer com que cada gerente tivesse uma chave do cofre, cuja abertura estaria sujeita ao controle de tempo. Durante o expediente, qualquer um dos dois gerentes poderia abrir o cofre, porém, fora do horário comercial, seria necessária a presença dos dois. Qual é a expressão lógica que habilita a abertura do cofre? Desenhe o circuito.

Tabela 2.3 | Horário de funcionamento da loja

T_2 (22h)	T_1 (10h)	
0	0	Antes do horário do expediente
0	1	Durante o expediente
1	0	-----
1	1	Após o expediente

Fonte: elaborada pela autora.

Resolução da situação-problema

O primeiro passo é montar a tabela-verdade com todas as possibilidades. Como a abertura do cofre está atrelada ao horário de funcionamento, temos que, para antes do horário do expediente e para depois do horário do expediente, são necessários os dois gerentes juntos (G_1 e G_2) para abrir o cofre. Durante o expediente, é necessário um **ou** outro gerente para abrir o cofre ($G_1 + G_2$).

Tabela 2.4 | Tabela-verdade, abertura do cofre

T_2 (22h)	T_1 (10h)	S (Abertura)
0	0	Dois gerentes (G_1G_2)
0	1	Um gerente ($G_1 + G_2$)
1	0	-----

1	1	Dois gerentes ($G_1 G_2$)
---	---	-----------------------------

Fonte: elaborada pela autora.

Com a tabela montada, retiramos a expressão lógica e fazemos a simplificação.

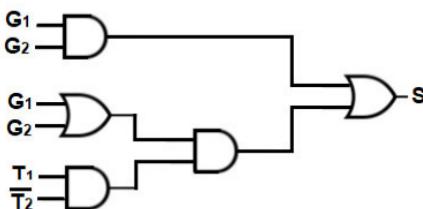
$$S = \overline{T_1 T_2} (G_1 G_2) + T_1 \overline{T_2} (G_1 + G_2) + T_1 T_2 (G_1 G_2)$$

$$S = G_1 G_2 (\overline{T_1 T_2} + T_1 T_2) + T_1 \overline{T_2} (G_1 + G_2)$$

$$S = G_1 G_2 (1) + T_1 \overline{T_2} (G_1 + G_2)$$

$$S = G_1 G_2 + T_1 \overline{T_2} (G_1 + G_2)$$

Figura 2.9 | Circuito digital, abertura do cofre



Fonte: elaborada pela autora.

Dessa forma, através do circuito digital, foi possível melhorar a segurança da relojoaria. Agora, você pode informar à loja que ela deverá efetuar a compra do circuito lógico implementado por você.

Faça valer a pena

1. Quantidades podem ser medidas, monitoradas, armazenadas, manipuladas, etc. Elas podem ser representadas de duas formas: analógica ou digital. As quantidades analógicas podem variar ao longo de uma faixa contínua de valores, e as digitais são representadas por símbolos.

Assinale a alternativa que apresenta corretamente apenas grandezas que estão relacionadas a quantidades digitais.

- Temperatura ambiente e termômetro de mercúrio.
- Grãos de areia de um parque e velocímetro de um automóvel antigo.
- Chave de cinco posições e corrente que flui em uma tomada elétrica residencial.

- d. Volume de água em um balde e subida usando uma rampa.
- e. Subida usando uma escada e quantidade de pedras em um vaso.

2. Podemos dividir os sistemas digitais em basicamente dois grupos: lógica combinacional e lógica sequencial. Quando as portas lógicas são interconectadas, elas podem produzir uma saída específica para as diferentes combinações possíveis das variáveis de entrada.

Assinale a alternativa que apresenta corretamente a expressão de saída simplificada de um circuito lógico com três entradas em que a saída só será nível alto quando a maioria das entradas forem nível baixo.

- a. $S = \bar{A}\bar{B} + \bar{B}\bar{C} + \bar{A}\bar{B}\bar{C}$
- b. $S = \bar{A}\bar{B}\bar{C} + \bar{B}\bar{C} + \bar{A}\bar{C}$
- c. $S = \bar{A}\bar{B} + \bar{B}\bar{C} + \bar{A}\bar{C}$
- d. $S = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C}$
- e. $S = \bar{A}\bar{B} + A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C}$

3. O termo “digital”, hoje em dia, virou rotina no nosso vocabulário. Isso se deve ao fato de os circuitos digitais e as técnicas digitais serem muito utilizadas em praticamente todas as áreas: computadores, automação e robótica, medicina e muitas outras. Ao tentar consertar o módulo eletrônico de um robô de brinquedo, foi verificada a existência de um circuito digital. Esse circuito tinha dois bits de entrada, A e B, e um bit de saída. Os bits A (menos significativos) e B eram utilizados para representar valores de números inteiros de 0 a 3 e, na prática, representavam os movimentos do robô. Após testes, verificou-se que a saída do circuito é 0 para os valores de entrada 0 e 2, e 1 para os valores de entrada 1 e 3.

Assinale a alternativa que apresenta corretamente a expressão que representa o circuito analisado.

- a. $S = B$
- b. $S = \bar{A}B$
- c. $S = AB$
- d. $S = A + AB$
- e. $S = A$

Seção 2

Circuitos combinacionais

Diálogo aberto

Olá! Vocês se lembram de quando vimos as características e as diferenças dos sistemas analógicos e dos sistemas digitais e de quando estudamos que os sistemas digitais podem ser divididos de duas maneiras: combinacional e sequencial? Esses conceitos serão importantes para continuarmos esse conteúdo tão desafiador.

No nosso dia a dia, estamos acostumados a manejar computadores e celulares, mas vocês sabem o que está por trás desses equipamentos? E nos cálculos por trás da nossa calculadora? E quando você esquece a porta do seu carro aberta e um som é emitido? Por trás de todos esses equipamentos estão os circuitos combinacionais. Sabia que até na entrada via *bluetooth* existe um circuito? Só que, nesse caso, o circuito é sequencial, pois se trata de registradores.

Deu para perceber quanto os circuitos combinacionais e sequenciais são importantes e estão presentes no nosso dia a dia?

Nesta seção veremos os circuitos combinacionais, aqueles cuja saída depende única e exclusivamente das possíveis combinações de entrada. Nesse contexto, podemos destacar os circuitos aritméticos, que são utilizados, principalmente, para construir a Unidade Lógica e Aritmética (ULA) dos microprocessadores e que estão presentes na nossa calculadora, no nosso computador, etc. Outros circuitos combinacionais importantes são os comparadores, os codificadores e os decodificadores. Por exemplo, usamos o decodificador para converter o sinal do processador para um formato que seja compatível com o display de sete segmentos, que há, por exemplo, na sua calculadora. Os circuitos codificadores e decodificadores transformam as informações obtidas de uma determinada maneira em informações em forma de código, as quais possam ser usadas por outros circuitos, por exemplo, o código Gray, que é utilizado em codificadores ópticos. Finalizaremos a seção com os MUX e DEMUX, sistemas digitais que podem processar informações de muitas formas, funcionando como conversores paralelo/série e série/paralelo. Um exemplo de DEMUX é o roteador, no qual temos um sinal de entrada e escolhemos por qual das saídas esse sinal passará.

Lembre-se de que você é o mais novo contratado por uma grande fábrica de brinquedos para ser o responsável por toda parte da eletrônica digital dos

brinquedos. O produto lançado para entreter as crianças foi o robô *Pet Dog*, um brinquedo que anda, tem sensores infravermelhos, late e tira fotos! Ele responde a comandos de voz e ao controle remoto.

O lançamento do *Pet Dog* foi um sucesso e, com isso, várias outras empresas resolveram lançar brinquedos semelhantes. Assim, a diretoria da empresa, juntamente com a equipe de marketing, solicitou que você criasse funções novas para o uso do controle remoto.

Uma delas é uma função especial inerente ao controle remoto, a qual reinicializa sua memória e suas preferências. Essa função é conseguida por uma combinação de teclas diferentes que, quando pressionadas em conjunto, iniciam um modo de reset da memória. Assim, seu novo trabalho é projetar um circuito codificador que lê uma combinação de seis teclas possíveis quando três delas em particular são pressionadas para dar um sinal para limpar a memória dos registradores.

Além disso, a equipe de marketing também pediu que você deixasse o *Pet Dog* mais atrativo, criando os chamados *Easter eggs* (ovos de Páscoa, em português), um conceito muito utilizado em jogos eletrônicos que remete a coisas escondidas, como comportamento programado, latidos diferentes e até mesmo canções. Segundo a equipe de marketing, a descoberta desses segredos despertaria um interesse maior dos usuários por seu funcionamento.

Para começar, podemos incluir um segredo: um latido de agradecimento com uma música, o qual deve ser chamado após uma caminhada de 15 minutos e após 2 minutos de proximidade com alguém.

Apresente a seu superior:

- A tabela-verdade.
- A expressão que executa a tabela.
- O circuito finalizado.

Os circuitos combinacionais e sequenciais estão presentes no nosso dia a dia. Só na sua casa, tenho certeza de que há vários equipamentos que possuem pelo menos uma combinação desses circuitos. Por esse motivo e pelo fato de os sistemas digitais estarem em constante evolução, é muito importante que você aprenda esse conteúdo e se dedique a ele. Bons estudos e tenha um ótimo trabalho!

Não pode faltar

Olá! Preparado para conhecer os circuitos combinacionais e suas aplicações? Os circuitos lógicos combinacionais são formados por uma combinação de portas lógicas, por meio das quais o nível lógico da saída é uma combinação dos níveis lógicos presentes nas entradas. Eles não possuem memória e suas saídas dependem exclusivamente dos valores atuais das entradas. Na prática, usamos um circuito lógico combinacional quando queremos solucionar problemas em que é necessária uma resposta, ou seja, quando há situações representadas pelas variáveis de entrada e, quando da sua implementação, é necessário obter as expressões da tabela-verdade, por exemplo o projeto de um controle de abertura da porta de um elevador.

Nesse âmbito, podemos elencá-los de acordo com suas funções, sendo que as principais são: somador, subtrator, comparador, codificador, decodificador, multiplexador e demultiplexador. Vamos conhecê-los?

Os circuitos aritméticos são sistemas digitais formados pela lógica booleana e são capazes de efetuar as quatro operações: soma, subtração, multiplicação e divisão. São utilizados, principalmente, para construir a ULA dos microprocessadores. Primeiramente, vamos estudar o meio somador (*half adder*), circuito que realiza a soma binária de um algarismo. Para facilitar esse estudo, vamos montar a tabela-verdade da soma de dois números binários de um algarismo (Tabela 2.5), na qual as entradas são representadas por A e a soma das entradas por e o transporte da saída por T_s (*carry out*).

Tabela 2.5 | Tabela-verdade do meio somador

A	B	S	T_s
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fonte: elaborada pela autora.

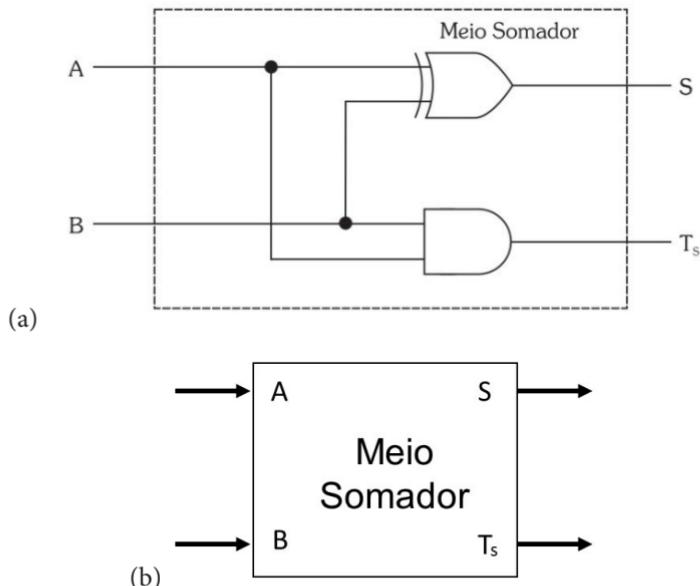
A partir da Tabela 2.5 podemos obter as expressões características do circuito para S e T_s :

$$S = \overline{A}\overline{B} + A\overline{B} = A\overline{A}$$

$$T_s = AB$$

A partir dessas equações podemos obter o circuito (Figura 2.10-a). O meio somador pode ser representado através de um bloco (Figura 2.10-b).

Figura 2.10 | Circuito do meio somador e bloco representativo



Fonte: Capuano (2014, p. 76-77).

O meio somador realiza a soma de números binários com apenas um algarismo. Para realizar a soma com mais algarismos, devemos considerar o transporte de entrada resultante da coluna anterior. O somador completo (*full adder*) efetua essa soma pois considera o transporte de entrada (*carry in*). Podemos apresentar a tabela-verdade do somador completo na tabela a seguir:

Tabela 2.6 | Tabela-verdade do somador completo

A	B	T _E	S	T _s
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0

0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fonte: elaborada pela autora.

A partir da Tabela 2.6 obtemos a expressão característica do circuito para S e simplificamos.

$$S = \overline{A}BT_E + \overline{ABT_E} + A\overline{BT_E} + ABT_E$$

$$S = \overline{A}(\overline{B}T_E + B\overline{T_E}) + A(\overline{B}\overline{T_E} + BT_E)$$

$$S = \overline{A}(B \oplus T_E) + A(B \odot T_E)$$

$$S = \overline{A}(B \oplus T_E) + A(\overline{B \oplus T_E})$$

$$X = B \oplus T_E$$

$$S = \overline{A}(X) + A(\overline{X})$$

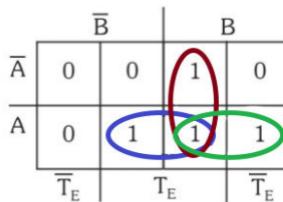
$$S = A \oplus X$$

$$S = A \oplus B \oplus T_E$$

Também obtemos a expressão para T_s e simplificamos pelo mapa de Karnough (Figura 2.11).

$$T_s = \overline{ABT_E} + A\overline{BT_E} + AB\overline{T_E} + ABT_E$$

Figura 2.11 | Mapa de Karnough



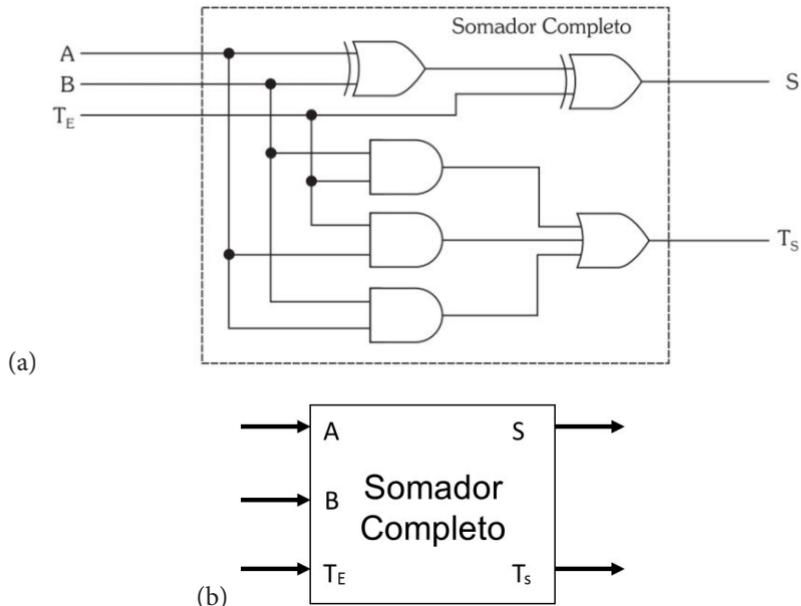
Fonte: adaptado de Capuano (2014).

A expressão simplificada possui três termos. E lembre-se: devemos obrigatoriamente inserir todos os uns na simplificação. Assim temos:

$$T_s = AT_E + BT_E + AB$$

Com as duas expressões simplificadas desenhamos o circuito do somador completo (Figura 2.12).

Figura 2.12 | Circuito somador completo (a) e bloco representativo (b)

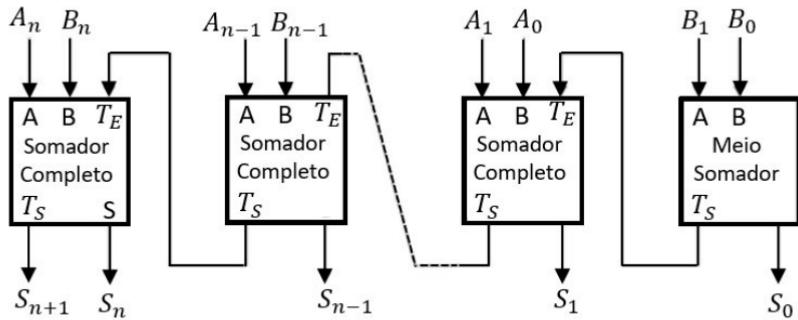


Fonte: Capuano (2014, p. 78).

Notamos que o somador completo é um circuito relativamente simples e que pode ser construído através de apenas seis portas lógicas.

Para efetuar a soma de dois números com n bits, montamos o diagrama de blocos (Figura 2.13). Observe que, a fim de somar os primeiros bits, utilizamos um meio somador, pois não há transporte de entrada.

Figura 2.13 | Diagrama de blocos de um somador completo



Fonte: adaptada de Capuano (2014).

Agora vamos estudar o meio subtrator (*half subtractor*), um circuito que realiza a subtração binária de um algarismo. Para facilitar seu estudo, vamos montar a tabela-verdade da subtração de dois números binários de um algarismo (Tabela 2.7), na qual as entradas são representadas por A e B, a subtração das entradas por e o transporte de saída por T_s (*carry out*).

Tabela 2.7 | Tabela-verdade do meio subtrator

A	B	S	T_s
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

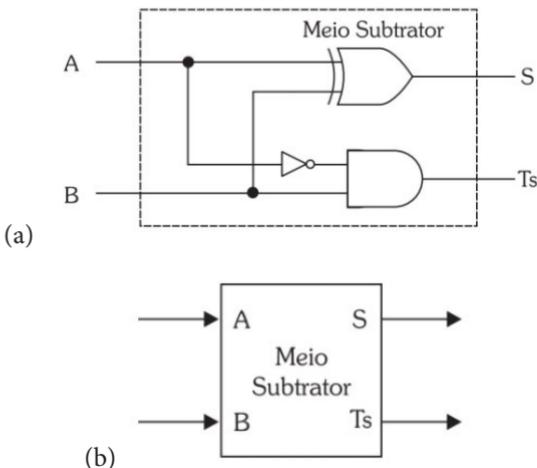
Fonte: elaborada pela autora.

A partir da Tabela 2.7 podemos obter as expressões características do circuito para S e T_s : $S = \overline{AB} + \overline{A}\overline{B} = A \oplus B$

$$T_s = \overline{AB}$$

A partir dessas equações podemos obter o circuito e seu bloco representativo (Figura 2.14).

Figura 2.14 | Circuito meio subtrator (a) e bloco representativo (b)



Fonte: Capuano (2014, p. 80).

Do mesmo modo que o meio somador, o meio subtrator realiza a subtração de números binários com apenas um algarismo. Para realizar a subtração com mais algarismos, devemos considerar o transporte de entrada resultante da coluna anterior. O subtrator completo (*full subtractor*) efetua essa subtração pois considera o transporte de entrada (*carry in*). A tabela-verdade do subtrator completo é apresentada na tabela adiante.

Tabela 2.8 | Tabela-verdade do subtrator completo

A	B	T_E	S	T_S
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Fonte: elaborada pela autora.

A partir da Tabela 2.8 obtemos a expressão característica do circuito para S. Como as saídas são iguais à do somador completo, temos:

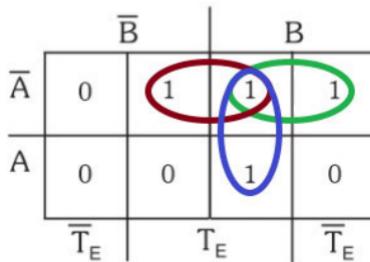
$$S = \overline{AB}T_E + \overline{A}\overline{B}T_E + A\overline{B}T_E + ABT_E$$

$$S = A \oplus B \oplus T_E$$

Também obtemos a expressão para T_E e simplificamos pelo mapa de Karnaugh (Figura 2.15).

$$T_S = \overline{AB}T_E + \overline{A}\overline{B}T_E + A\overline{B}T_E + ABT_E$$

Figura 2.15 | Mapa de Karnaugh

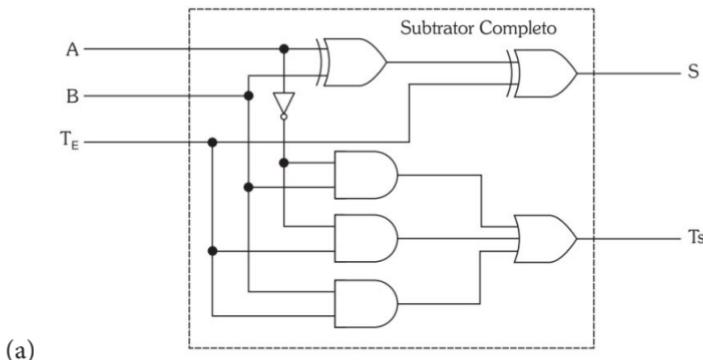


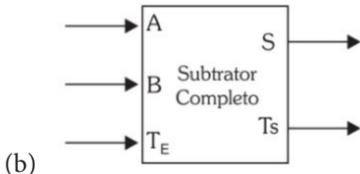
Fonte: adaptada de Capuano (2014).

$$T_S = \overline{A}T_E + BT_E + \overline{AB}$$

Com as duas expressões simplificadas, desenhamos o circuito do subtrator completo e seu bloco representativo (Figura 2.16).

Figura 2.16 | Subtrator completo (a) e bloco representativo (b)

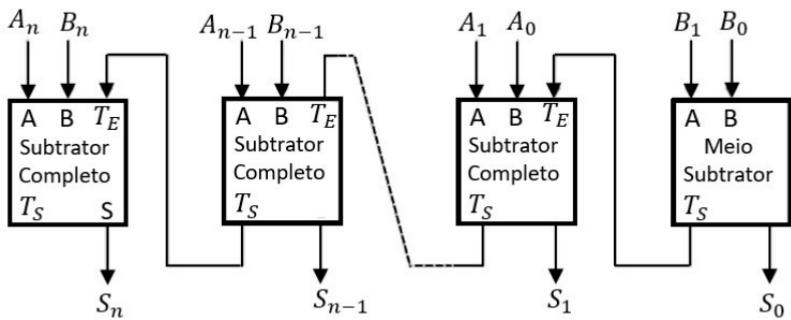




Fonte: Capuano (2014, p. 80-81).

Do mesmo modo que o somador, também podemos esquematizar um sistema subtrator para dois números de n bits (Figura 2.17).

Figura 2.17 | Diagrama de blocos para um subtrator genérico de dois números de n bits



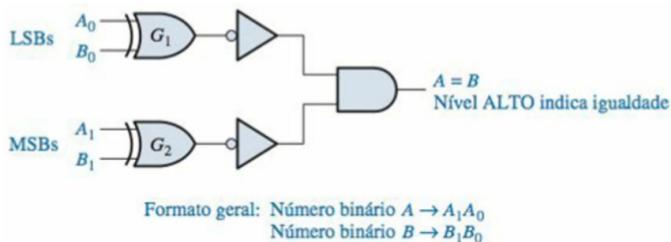
Fonte: adaptada de Capuano (2014).

Um comparador faz a comparação das magnitudes de dois números binários para determinar se são iguais.

Uma porta lógica que faz essa comparação é a EX-OR, pois sua saída é nível 1 se os dois bits de entrada forem diferentes e é 0 se os bits de entrada forem iguais.

A comparação de dois números binários de 2 bits pode ser representada através de portas lógicas (Figura 2.18). Notamos que os dois bits menos significativos (LSBs) dos dois números são comparados pela porta EX-OR G_1 e os dois bits mais significativos (MSBs) dos dois números são comparados pela porta EX-OR G_2 .

Figura 2.18 | Circuito comparador

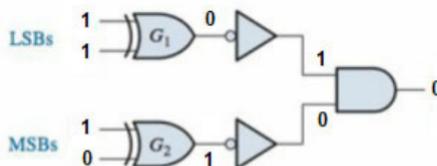


Fonte: Floyd (2007, p. 328).

Exemplificando

Vamos agora exemplificar como é determinada a saída do circuito comparador (Figura 2.19) para as entradas 01 e 11.

Figura 2.19 | Exemplo de um circuito comparador



Fonte: adaptada de Floyd (2007).

A saída do comparador é 0, portanto os números são diferentes.

O decodificador é um tipo de circuito digital que identifica a presença de um código em suas entradas e, através de um nível de saída predeterminado, indica a presença desse código.

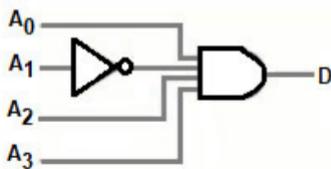
Exemplificando

Vamos agora exemplificar como se determina a lógica para decodificar o número binário 1101 produzindo um nível ALTO na saída.

A função do decodificador pode ser encontrada complementando as variáveis que aparecem como 0 no número binário desejado:

$D = A_3A_2\overline{A}_1A_0$. A função pode ser implementada através de uma porta AND e de uma porta NOT (Figura 2.20).

Figura 2.20 | Decodificador



Fonte: elaborada pela autora.

O decodificador Decimal/8421 converte um código decimal em código BCD (8421). A entrada do código decimal é feita por um conjunto de chaves numeradas de 0 a 9, e a saída, através de quatro fios, fornecendo um código binário de 4 bits, correspondente à chave selecionada. Podemos ver, na tabela a seguir, as funções de decodificação para os dez códigos BCD.

Tabela 2.9 | Funções de decodificação para o código BCD

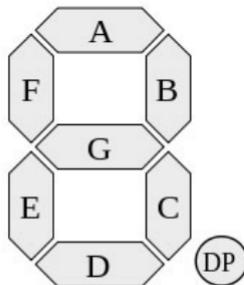
Digito Decimal	Código BCD				Função de decodificação
	A	B	C	D	
0	0	0	0	0	\overline{ABCD}
1	0	0	0	1	$\overline{ABC}\bar{D}$
2	0	0	1	0	$\overline{AB}\overline{CD}$
3	0	0	1	1	$\overline{A}\overline{BCD}$
4	0	1	0	0	$\overline{A}\overline{B}\overline{C}\bar{D}$
5	0	1	0	1	$\overline{A}\overline{B}\bar{C}\bar{D}$
6	0	1	1	0	$\overline{A}\bar{B}\overline{C}\bar{D}$
7	0	1	1	1	$\overline{A}\bar{B}CD$
8	1	0	0	0	$A\overline{B}\overline{C}\overline{D}$
9	1	0	0	1	$A\overline{B}\overline{C}\bar{D}$

Fonte: elaborada pela autora.

Cada uma dessas dez funções é implementada com portas NAND para obter saídas ativas em nível BAIXO, porém, caso precise de uma saída ativa em nível ALTO, são usadas portas AND.

O decodificador de BCD para sete segmentos recebe o código BCD na entrada e fornece saídas que acionam um display de sete segmentos (Figura 2.21), que faz a leitura decimal de números de 0 a 9 e alguns símbolos que podem ser letras ou sinais.

Figura 2.21 | Display de sete segmentos



Fonte: Wikimedia Commons .

Os leds podem ser de dois tipos: catodo comum e anodo comum. O tipo mais comum é o catodo comum, que acende quando é aplicado o nível 1. O anodo comum acende quando é aplicado o nível 0.

Para se escrever a sequência de 0 a 9 em um display de sete segmentos de um catodo comum, a partir do código BCD 8421, vamos começar verificando os segmentos que devem ser acesos e atribuir o nível 1 a eles. A sequência de caracteres do decodificador BCD8421/7 segmentos é representada na figura (Figura 2.22).

Observe, na Figura 2.22, que, para acender o número zero, devemos colocar nível alto nos segmentos “a”, “b”, “c”, “d”, “e” e “f” e, para acender

o número sete, os segmentos “a”, “b” e “c” devem estar em nível alto. Para acender os demais números, devemos seguir a mesma lógica.

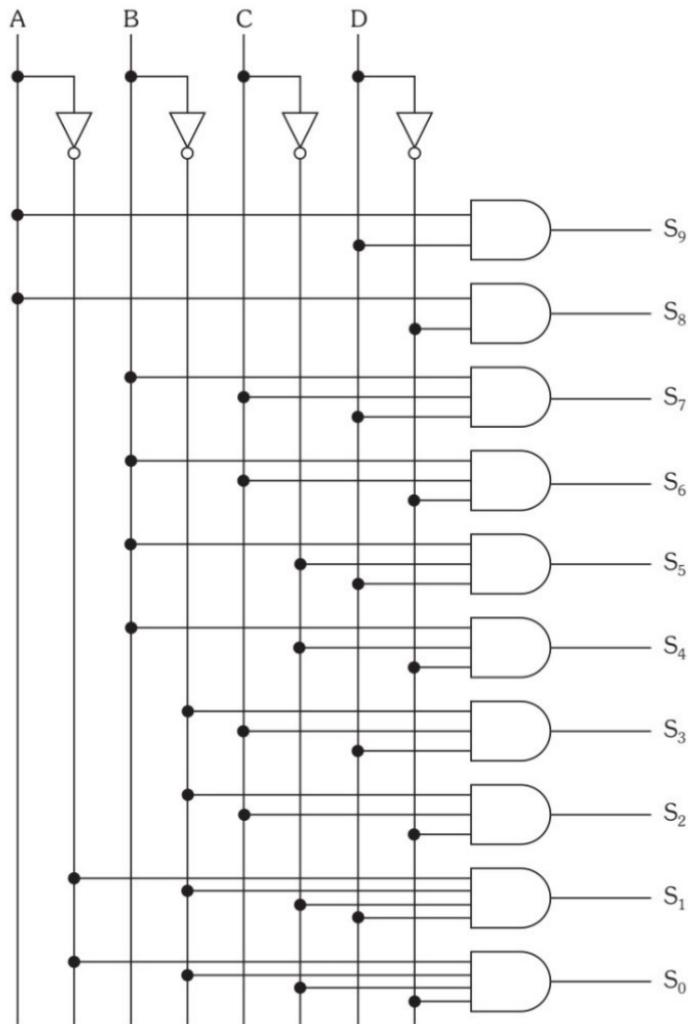
Agora, podemos simplificar cada segmento através do mapa de Karnough e os números que não fazem parte da sequência consideraremos irrelevantes (Figura 2.23).

Figura 2.22 | Sequência de caracteres do decodificador BCD8421/7 segmentos

Caracteres	Display	BCD 8421				Código para 7 segmentos						
		A	B	C	D	a	b	c	d	e	f	g
0		0	0	0	0	1	1	1	1	1	1	0
1		0	0	0	1	0	1	1	0	0	0	0
2		0	0	1	0	1	1	0	1	1	0	1
3		0	0	1	1	1	1	1	1	0	0	1
4		0	1	0	0	0	1	1	0	0	1	1
5		0	1	0	1	1	0	1	1	0	1	1
6		0	1	1	0	1	0	1	1	1	1	1
7		0	1	1	1	1	1	1	0	0	0	0
8		1	0	0	0	1	1	1	1	1	1	1
9		1	0	0	1	1	1	1	1	0	1	1

Fonte: Capuano (2014, p. 72).

Figura 2.23 | Circuito decodificador simplificado



Fonte: Capuano (2014, p. 71).

Saiba mais

O display de sete segmentos pode escrever outros caracteres que também são muito utilizados em sistemas digitais para representar outras funções e palavras. Na sequência, a figura representa outras possibilidades de caracteres.

Figura 2.24 | Caracteres em um display de sete segmentos

A	b	C c	d	E e	F
G g	H h	I i	J	L	M
n	O o	P	Q	r	S
t	u	U	Y	-	D

Fonte: Capuano (2014, p. 75).

Refletá

Decodificadores são usados em computadores para seleção de entrada/saída, pois eles têm que se comunicar com muitos periféricos enviando e/ou recebendo dados através das portas de entrada/saída (I/O).

Um codificador é um circuito que realiza a função inversa do decodificador e pode ser implementado para codificar símbolos e caracteres alfabéticos. Um codificador decimal/BCD possui dez entradas, ou seja, uma para cada dígito decimal, e quatro saídas, que são as do código BCD (Tabela 2.10). A partir da Tabela 2.10 montamos o circuito decodificar decimal/binário (Figura 2.25).

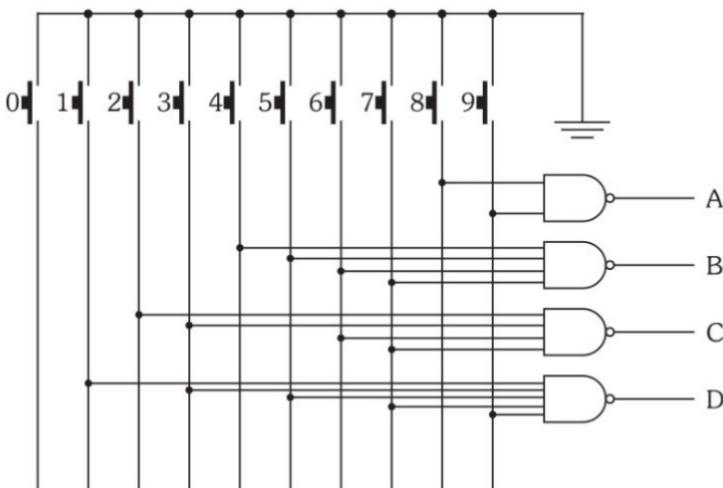
Tabela 2.10 | Codificador Decimal/BCD

Dígito Decimal	Código BCD			
	A	B	C	D
0 (CH0)	0	0	0	0
1 (CH1)	0	0	0	1
2 (CH2)	0	0	1	0
3 (CH3)	0	0	1	1
4 (CH4)	0	1	0	0
5 (CH5)	0	1	0	1
6 (CH6)	0	1	1	0

7 (CH7)	0	1	1	1
8 (CH8)	1	0	0	0
9 (CH9)	1	0	0	1

Fonte: elaborada pela autora.

Figura 2.25 | Circuito codificador decimal/binário



Fonte: Capuano (2014, p. 69).

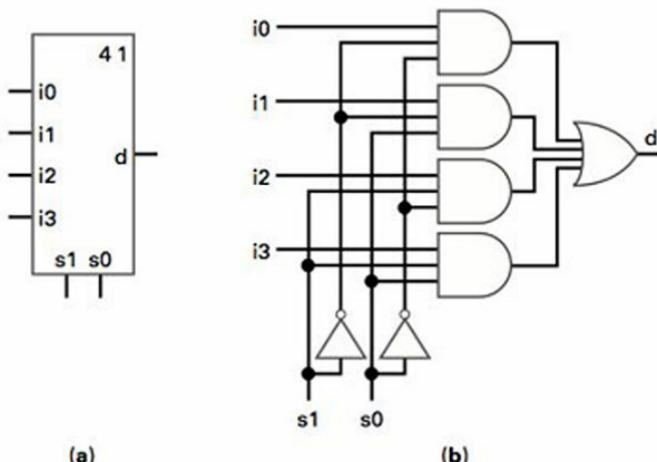
Nota-se que a chave CH0 não está ligada a nenhuma das entradas das portas, ou seja, ela é irrelevante. Um exemplo de aplicação dos codificadores e decodificadores são as calculadoras.

Um multiplexador (ou MUX) ou seletor de dados é um dispositivo $M \times 1$ que possui M entradas de dados e 1 saída, isso significa que apenas uma das informações digitais das entradas é selecionada para ir à saída. O MUX possui entradas de seleção de dados, e são essas entradas que permitem que os dados digitais selecionados de quaisquer entradas sejam comutados a saída.

Uma das aplicações do MUX é a geração de funções lógicas combinacionais na forma de soma de produtos (VAHID, 2008).

Um multiplexador de quatro entradas 4×1 (Figura 2.26), possui quatro entradas de dados I_3 , I_2 , I_1 e I_0 , duas entradas de seleção s_1 e s_0 , e uma saída de dados.

Figura 2.26 | Multiplexador 4x1: bloco (a) e circuito (b)

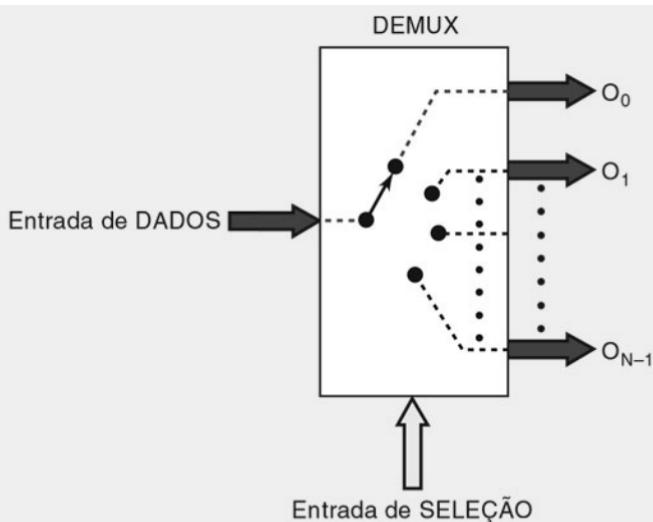
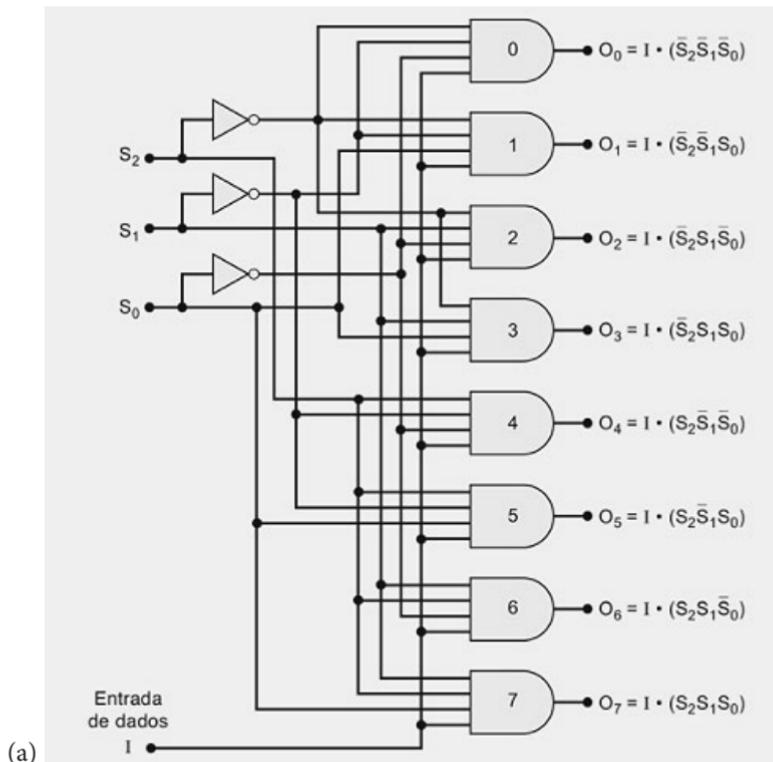


Fonte: Vahid (2008, p. 96).

Um demultiplexador, DEMUX, ou distribuidor de dados (Figura 2.27) inverte a função da multiplexação e possui a forma $1 \times M$, ou seja, tem uma entrada de dados e , com base nos valores de $\log_2(M)$ linhas de seleção, pega essa informação e passa para uma das M saídas disponíveis; as demais ficam em zero.

No DEMUX 1×8 (Figura 2.27), a informação I (entrada) é conectada nas oito portas AND, porém apenas uma será habilitada pelas entradas de seleção S_0 , S_1 e S_2 . Por exemplo, se a entrada de seleção for 011, a informação irá para a saída O_3 .

Figura 2.27 | Circuito multiplexador (a) e bloco demultiplexador generalizado (b)



Fonte: Tocci, Widmer e Moss (2011, p. 531-530).

Algumas aplicações do DEMUX são realizadas em circuitos, que devem receber uma informação digital, e para conversão de informação serial em paralela.

Nesta seção, nós vimos mais profundamente os circuitos combinacionais e suas aplicações. Esses circuitos são aqueles nos quais a saída depende apenas das combinações atuais das entradas, portanto não possuem memória. Iniciamos a seção vendo o meio somador e o meio subtrator, pois assim teríamos uma base para construir o circuito somador e subtrator completo. Continuamos a seção através do entendimento e da compreensão dos circuitos comparador, codificador e decodificador e finalizamos com os circuitos multiplexador e demultiplexador.

Na próxima seção, veremos os circuitos sequenciais, que são aqueles que possuem memória. Veremos também os *flip-flops*, os contadores e registradores, o que é uma máquina de estado e os temporizadores.

Bons estudos!

Sem medo de errar

Quanto à situação-problema apresentada no início desta seção, primeiramente, vamos montar a tabela-verdade, que nada mais é do que as saídas de um sistema para todas as combinações possíveis de entrada. Através dessa tabela, é possível definir se a proposição é verdadeira ou falsa, mas, para isso, vamos definir quantas combinações teremos nela. Como o circuito codificador vai ler uma combinação de seis teclas, nós teremos seis entradas: e F. Três dessas em particular serão pressionadas a dar um sinal para limpar a memória dos registradores, logo temos uma saída que é o RESET. Dessa forma, a saída só será atividade quando existir a combinação de três números 1, ou seja, quando estiverem ativadas (Tabela 2.11).

Tabela 2.11 | Combinações possíveis

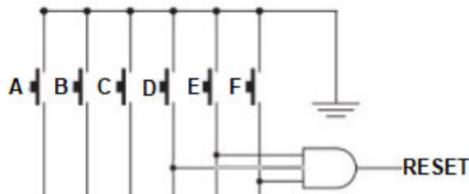
A	B	C	D	E	F	RESET
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	0	1	1	0
0	0	0	1	0	0	0
0	0	0	1	1	1	1

Fonte: elaborada pela autora.

Nesse caso, a única combinação possível será: $RESET = DEF$.

O segundo passo é projetar o circuito. Nesse caso, usamos uma porta lógica AND de três entradas, D, E e F, pois são as únicas que possuem 1 ao mesmo tempo (Figura 2.28).

Figura 2.28 | Circuito codificador decimal/binário



Fonte: elaborada pela autora.

Agora vamos desenvolver o circuito do segredo, lembrando que, ao latir em agradecimento, o *Pet Dog* deve fazê-lo com uma música, o que deve acontecer após uma caminhada de 15 minutos e após 2 minutos de proximidade com alguém.

Primeiramente vamos montar a tabela-verdade. Com essa tabela é possível definir se a proposição é verdadeira ou falsa, mas, para isso, vamos definir quantas combinações teremos nela. O nosso *Pet Dog* tem dois sensores: um de tempo e outro de proximidade, portanto teremos quatro entradas: +15 minutos, -15 minutos, +2 minutos e -2 minutos. Devemos ficar atentos aos casos que são impossíveis de acontecer, como andar mais de 15 minutos e menos de 15 minutos ao mesmo tempo e ficar mais de 2 minutos e menos de 2 minutos próximos.

Tabela 2.12 | Combinações possíveis

- 15 min.	+ 15 min.	- 2 min.	+ 2 min.	Latido com música
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	Impossível
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	Impossível
1	0	0	0	0

1	0	0	1	0
1	0	1	0	0
1	0	1	1	Impossível
1	1	0	0	Impossível
1	1	0	1	Impossível
1	1	1	0	Impossível
1	1	1	1	Impossível

Fonte: elaborada pela autora.

Observando a tabela vemos que o único caso possível é: latido_musica=+15 min. 2 min.

O circuito é formado com apenas uma porta AND (Figura 2.29).

Figura 2.29 | Porta AND



Fonte: elaborada pela autora.

Dessa forma, foi possível entregar ao cliente as informações solicitadas, sendo: a tabela-verdade, a expressão que executa a tabela e o circuito finalizado.

Avançando na prática

Modernizando a fila de espera de um restaurante

Após um grande aumento no número de clientes de um restaurante, você foi contratado para modernizar a fila de espera. No modelo atual, quando um cliente chega ao restaurante, ele recebe da recepcionista um número em papel e, quando chega a sua vez de entrar no local, um funcionário o chama pelo número.

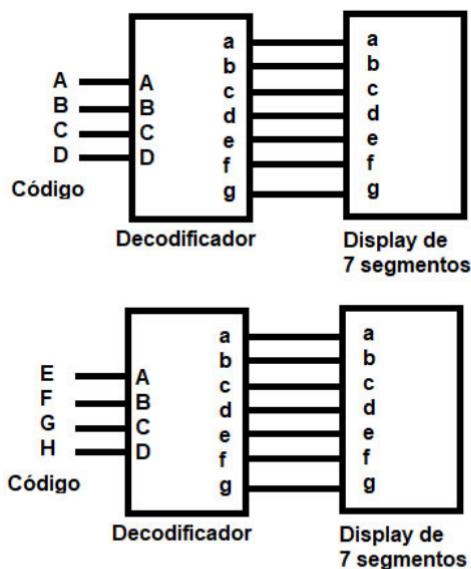
O dono do restaurante solicitou que você desenvolvesse um sistema em que o cliente receba um código de 8 bits via celular. Esse código gerará a

posição dele na fila de espera e deverá aparecer em um display na frente do restaurante quando chegar a vez dele de ser atendido.

Resolução da situação-problema

Após uma revisão dos conceitos de circuitos combinacionais, você verificou que, para solucionar o problema, basta utilizar dois decodificadores e dois displays de sete segmentos. Como o código que o usuário recebe é formado por 8 bits, ou seja, duas palavras de quatro, podemos representá-lo por A, B, C e D em um decodificador e E, F, G e H no outro decodificador. Esses bits são codificados em um código de sete bits que acenderão os LEDs (em um display que mostra o número decimal da posição do cliente na fila para ser atendido). A Figura 2.30 apresenta o esquema.

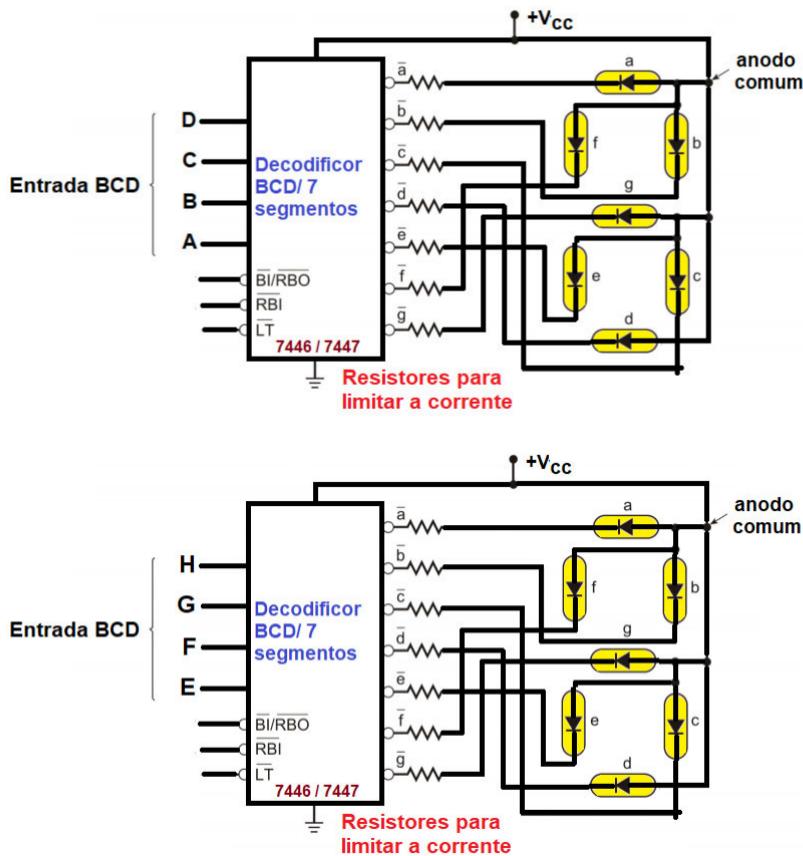
Figura 2.30 | Ligação dos decodificadores no display de sete segmentos



Fonte: elaborada pela autora.

Comercialmente pode-se usar dois CIs 7446 ou 7447 e dois displays de sete segmentos (Figura 2.31)

Figura 2.31 | Ligação dos Cls no display de sete segmentos



Fonte: elaborada pela autora.

Dessa forma, você consegue implementar o sistema solicitado pelo cliente.

Faça valer a pena

1. Um multiplexador (*multiplexer*) é um dispositivo utilizado para selecionar um sinal de entrada analógico ou digital, para que apenas este vá para a saída.

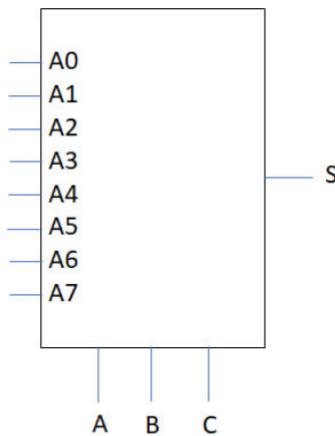
Na maioria dos casos, um multiplexador pode ter até 2^N entradas e necessita de N entradas de seleção.

Um multiplexador 4x1 tem os seguintes valores de entrada: $I_0 = 0$, $I_1 = 1$, $I_2 = 1$ e $I_3 = 0$. Assinale a alternativa que apresenta corretamente o valor da saída de um multiplexador para o valor de entrada de seleção: 01.

- a. 0.
- b. 1.
- c. 00.
- d. 11.
- e. 01.

2. O multiplexador é aplicado nos casos em que necessitamos enviar várias informações, que estão contidas em vários canais, a apenas um canal. A figura representa um multiplexador de oito entradas.

Figura | Multiplexador de oito entradas



Fonte: elaborada pela autora.

Através do multiplexador, é possível implementar $X = ABC + \bar{A}C + B\bar{C}$, o que está associado à saída S e às entradas de controle A, B e C. Assinale a alternativa que apresenta corretamente o estado das entradas A0, A1, A2, A3, A4, A5, A6 e A7.

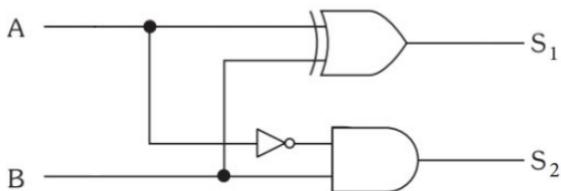
- a. A0=1; A1=1; A2=1; A3=1; A4=1; A5=1; A6=1 e A7=1.
- b. A0=0; A1=0; A2=1; A3=0; A4=0; A5=1; A6=1 e A7=1.
- c. A0=0; A1=1; A2=1; A3=1; A4=0; A5=0; A6=1 e A7=1.

- d. $A_0=1; A_1=0; A_2=1; A_3=0; A_4=0; A_5=1; A_6=0$ e $A_7=0$.
- e. $A_0=1; A_1=0; A_2=0; A_3=1; A_4=1; A_5=0; A_6=1$ e $A_7=1$.

3. Circuitos combinacionais são aqueles cuja saída depende única e exclusivamente das possíveis combinações de entrada. Nesse contexto, podemos destacar os circuitos aritméticos, os comparadores, os codificadores, os decodificadores, os multiplexadores e os demultiplexadores.

Assinale a alternativa que apresenta corretamente como o circuito representado na figura pode ser utilizado.

Figura | Exemplo de circuito hipotético



Fonte: adaptada de Capuano (2014).

- a. Somador.
- b. Meio subtrator.
- c. Multiplexador.
- d. Decodificador.
- e. Comparador.

Seção 3

Flip-flops e circuitos correlatos

Diálogo aberto

Olá! Você está gostando de aprender sobre o universo dos circuitos lógicos e combinacionais? É muito interessante, não acha? Agora, vamos aprender sobre os circuitos que lembram. Vocês devem estar pensando: e desde quando circuitos se lembram de alguma coisa? Vamos à explicação.

Os circuitos sobre os quais aprenderemos são os sequenciais, ou seja, os que possuem memória, mas não é aquela memória em que os dados, as instruções de programa e outras coisas são armazenadas. A memória de que estou falando fica dentro da CPU e é chamada de registrador. De uma forma bem simples, registrador é como um bloquinho de anotações descartável, ou seja, no nosso mundo digital, os registradores são usados para armazenar valores temporários e, com isso, podemos usar uma memória anterior em futuras operações.

Ainda está muito abstrato? Conhece aquelas máquinas de salgadinhos, nas quais você vai colocando as moedas até atingir o valor do produto? Então, esse é um tipo de registrador, pois, quando colocamos uma moeda e depois outra e outras, a máquina lembra do resultado da primeira moeda para somar com a segunda e com as demais e depois dá o resultado. Não é incrível?

O responsável pela construção dos registradores e dos contadores e por outras lógicas de controle sequencial é o *flip-flop*, que é implementado a partir das portas lógicas.

A maioria dos circuitos digitais funcionam de um modo sincronizado e isso é conseguido através de sinais retangulares, que são gerados por um oscilador. Este produz o sinal de *clock* que deve ser perfeitamente retangular e possuir uma frequência de acordo com o circuito a ser sincronizado. Um oscilador é um gerador de forma de onda autossustentado e é usado como fonte para formas de onda de temporização em sistemas digitais.

Lembre-se de que você é o mais novo contratado por uma famosa fábrica de brinquedos para ser responsável por toda parte da eletrônica digital dos brinquedos. O produto lançado para entreter as crianças foi o robô *Pet Dog*, um brinquedo que anda, tem sensores infravermelhos, late e tira fotos! Ele responde a comandos de voz e ao controle remoto.

O lançamento do *Pet Dog* foi um sucesso e, com isso, várias outras empresas resolveram lançar brinquedos semelhantes. Assim, a diretoria da empresa, juntamente com a equipe de marketing, solicitou que você criasse funções novas para o uso do controle remoto.

Uma delas é a função especial inerente ao controle remoto, a qual re inicializa sua memória e suas preferências; e a outra, que deixou o *Pet Dog* mais atrativo, foi a criação dos chamados *Easter eggs* (ou ovos de Páscoa, em português), por meio da qual foi implementado um latido de agradecimento com uma música que deve ser chamada após uma caminhada de 15 minutos e de 2 minutos de proximidade com alguém.

A diretoria da fábrica está cada vez mais satisfeita com o resultado das vendas do *Pet Dog* e pediu que você fizesse uma ampliação da capacidade de armazenamento das fotos do brinquedo. Inicialmente, o *Pet Dog* foi lançado para armazenar fotos, mas não há nenhum aviso sobre quantas fotos já foram batidas. O seu trabalho agora será projetar um circuito contador que conte até cinco.

Suponha que já exista um circuito separado com câmera e memória e, a partir disso, surgem os seguintes pontos que precisam ser contemplados no projeto: a tabela-verdade do contador; a expressão que executa a tabela; o circuito do contador, utilizando o *flip-flop T*; e o circuito do contador finalizado (ligação com a memória).

E então, está pronto para aprender? Continue estudando e não desanime, pois esse conteúdo é de extrema importância para você, afinal os sistemas digitais estão sempre presentes no nosso dia a dia! Bons estudos e tenha um ótimo trabalho!

Não pode faltar

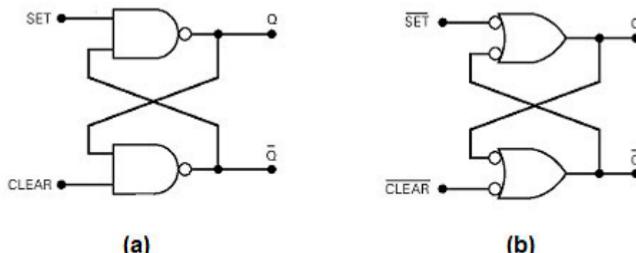
Como já se sabe, o computador só entende 1s e 0s. Dessa forma, o conceito de memória, para o computador, consiste em armazenar vários 1s (nível alto) e 0s (nível baixo), de maneira que, para armazenar 1s, precisamos pensar em um modo de manter esse nível alto, ou seja, reter esse estado por um tempo maior (processo denominado *latch*), sendo também necessário armazenar os zeros e, preferencialmente, cada bit separado.

Para isso, existem os circuitos chamados de *flip-flops*, que possuem dois estados estáveis (também chamados de biestáveis), possibilitando dividir frequências e deslocar palavras binárias, tendo como principais aplicações os contadores e registradores (CAPUANO, 2014).

Como existem vários tipos de *flip-flops*, o símbolo das entradas mudam, diferentemente da saída, que sempre é representada pelo mesmo, sendo este o Q , o qual permanecerá no estado 1 ou 0, e o \bar{Q} , que sempre será o inverso de Q (se Q for 0, \bar{Q} é 1). Os circuitos do tipo *flip-flop* mudam de acordo com sua função, podendo ser do tipo RS (*Reset Set*), D, T e JK. Vamos conhecê-los?

O primeiro e mais simples *flip-flop* que vamos ver é o RS (*Reset Set*) ou *latches*, o qual pode ser constituído a partir de duas portas NAND (Figura 2.32 (a)) ou de duas portas NOR (Figura 2.32 (b)).

Figura 2.32 | Exemplo de *flip-flop* RS composto de duas portas NAND (a) e duas portas NOR (b)



Fonte: adaptada de Tocci, Widmer e Moss (2011).

Podemos analisar as suas saídas Q e \bar{Q} montando uma tabela com os valores possíveis de entrada (Tabela 2.13). Quando o SET e o CLEAR (RESET) estão em zero, chamamos essa situação de condição proibida, pois não é possível dispará-los ao mesmo tempo já que, se esse caso acontecesse, teríamos a saída Q e \bar{Q} iguais, o que é impossível de acontecer (CAPUANO, 2014).

Tabela 2.13 | *Flip-flop* tipo RS

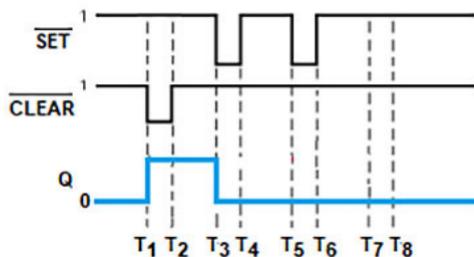
SET	CLEAR (RESET)	Q	\bar{Q}
0	0	Condição proibida	
0	1	0	1
1	0	1	0
1	1	Não muda (mantém a condição anterior)	

Fonte: elaborada pela autora.

Exemplificando

Independentemente das entradas do *flip-flop*, quando o SET está ativo, o *flip-flop* é, obrigatoriamente, um, enquanto, se o RESET estiver ativo, o *flip-flop* é, obrigatoriamente, zero. Dessa forma, considerando inicialmente $Q=0$ em um *flip-flop RS*, podemos determinar a forma de onda da saída como:

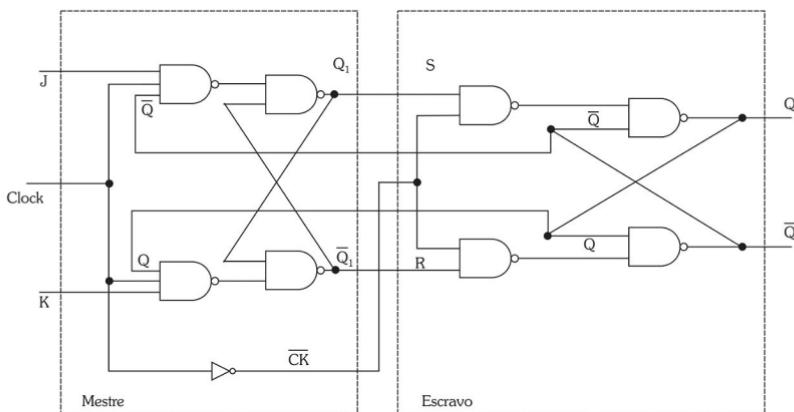
Figura 2.33 | Exemplo de entrada e saída do tipo *flip-flop*



Fonte: elaborada pela autora.

Como vimos, o *flip-flop RS* apresenta uma condição proibida quando as entradas R e S assumem o valor 0 simultaneamente. Mas como corrigir essa condição? Para isso, podemos utilizar o *flip-flop JK* mestre-escravo (JK Master-Slave) (Figura 2.34), que resolve esse problema.

Figura 2.34 | *Flip-flop JK* mestre-escravo

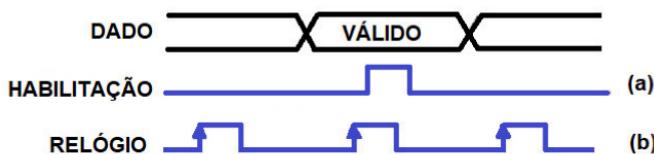


Fonte: Capuano (2014, p. 90).

Antes de aprofundarmos as explicações sobre o *flip-flop* JK, vamos ver um pouco sobre os tipos de entrada de *clock* existentes nos *flip-flops*.

Há dois tipos de entrada de *clock* em *flip-flops*: sensível ao estado e sensível à borda (que pode ser de subida ou de descida) e, apesar de serem utilizados para o mesmo fim, os circuitos de temporização são diferentes. No *flip-flop* com entrada de *clock* sensível ao estado, a memorização ocorrerá durante o estado alto do sinal de *clock* (Figura 2.35 (a)). Já nos *flip-flops* com entrada de relógio sensível à borda de subida, a memorização ocorrerá no instante da transição de descida ou de subida do sinal de *clock* (Figura 2.35 (b)) (CAPUANO, 2014).

Figura 2.35 | Tipos de entrada de *clock* em *flip-flops*



Fonte: elaborada pela autora.

O *flip-flop* JK mestre-escravo é o tipo de *clock* sensível à borda de descida (o *clock* é ativo quando passa de 1 para 0) e está representado através da Figura 2.36 (a). O mesmo bloco também pode representar o *clock* sensível à borda de subida (o *clock* é ativo quando passa de 0 para 1) (Figura 2.36 (b)).

Figura 2.36 | Entrada *clock* sensível à borda de descida (a) e entrada *clock* sensível à borda de subida (b)



Fonte: elaborada pela autora.

A saída *Q* vai assumir valores, conforme a situação das entradas JK, somente após a passagem do *clock* de acordo com o tipo, sensível à borda de subida ou de descida. Podemos analisar a saída *Q* do circuito *flip-flop* JK mestre-escravo mencionado montando uma tabela com os valores possíveis de entrada (Tabela 2.14).

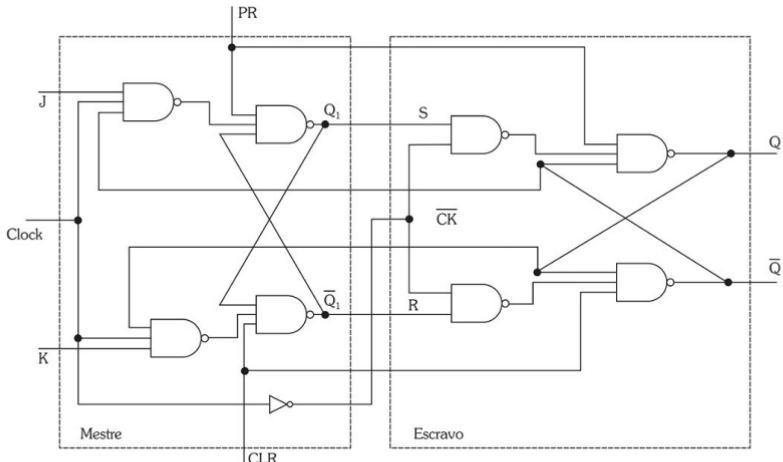
Tabela 2.14 | *Flip-flop* JK mestre-escravo

J	K	Q_f
0	0	Q_a
0	1	0
1	0	1
1	1	$\overline{Q_a}$

Fonte: elaborada pela autora.

O *flip-flop* JK, através das entradas PRESET (PR) e CLEAR ou RESET (CLR), pode assumir os valores de saída $Q=1$ ou $Q=0$, respectivamente. Podemos representar o *flip-flop* JK mestre-escravo com as entradas PRESET e CLEAR ativas em 0 (Figura 2.37) (CAPUANO, 2014).

Figura 2.37 | JK mestre-escravo com PRESET e CLEAR



Fonte: Capuano (2014, p. 92).

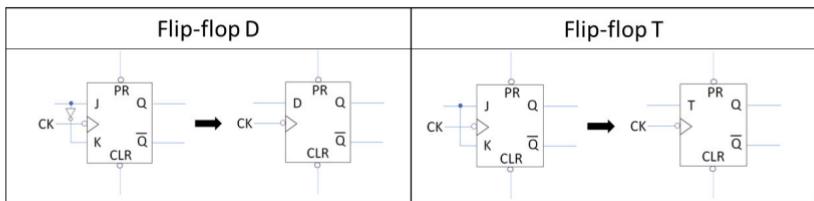
Assimile

As entradas CLEAR e PRESET não podem assumir o valor 0 simultaneamente, pois isso levaria à saída uma situação não permitida; porém, quando assumem o valor 1, o funcionamento é normal.

A partir do *flip-flop* JK mestre-escravo com a entrada K invertida, ou seja, colocando-se um inverter, é possível construir o *flip-flop* tipo D (Figura 2.38). Podemos analisar a saída Q do circuito *flip-flop* D montando a tabela-

verdade com os valores possíveis de entrada (Tabela 2.15). Notamos que J e K nunca poderão ter valores iguais na configuração de *flip-flop* D, pois as entradas estão curto-circuitadas (CAPUANO, 2014). Se curto-circuitarmos as entradas JK do *flip-flop* mestre-escravo (ligar uma na outra), obteremos o *flip-flop* T (*Toggle* – comutado). Podemos analisar a saída Q do circuito *flip-flop* T montando a tabela-verdade com os valores possíveis de entrada (Tabela 2.15).

Figura 2.38 | *Flip-flop* D obtido a partir do JK mestre-escravo e do bloco representativo; *flip-flop* T obtido a partir do JK mestre-escravo e do bloco representativo



Fonte: elaborada pela autora.

Tabela 2.15 | Tabela do *flip-flop* D obtido a partir do JK mestre-escravo e do *flip-flop* T obtido a partir do JK mestre-escravo

<i>Flip-flop</i> D			<i>Flip-flop</i> T
J	K	D	Q_f
0	0	Impossível	/
0	1	0	0
1	0	1	1
1	1	Impossível	/
J	K	T	Q_a
0	0	0	0
0	1	Impossível	/
1	0	1	1
1	1	1	$\overline{Q_a}$

Fonte: elaborada pela autora.

Notamos que J e K nunca poderão ter valores diferentes na configuração de *flip-flop* T, pois as entradas estão curto-circuitadas. O fato do *flip-flop* T, com a entrada T igual a 1, complementar a saída Q_a a cada descida de *clock*, faz com que ele seja utilizado nos contadores assíncronos.

Contadores nada mais são que circuitos digitais que variam os seus estados de saída de acordo com o pulso de *clock* e com uma sequência de contagem já predeterminada no projeto, podendo eles serem assíncronos e síncronos.

Nos contadores assíncronos, os *flip-flops* funcionam de maneira assíncrona (sem sincronismo), o que significa que eles não possuem a mesma entrada de

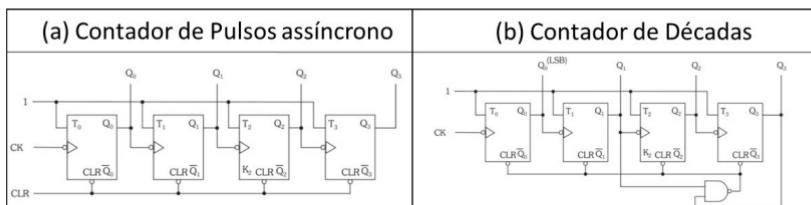
clock. Nele, o *clock* fica apenas no primeiro *flip-flop*, sendo as entradas dos demais conectadas às saídas *Q* dos respectivos antecessores e, nos demais, são derivadas das saídas dos *flip-flops* anteriores (CAPUANO, 2014).

Um tipo de assíncrono é o contador de pulsos, o qual apresenta, nas saídas, o sistema binário em sequência. Ele é formado por quatro *flip-flops* T ou JK mestre-escravo, com a entrada T, ou J e K iguais a 1, que gera na saída $Q_f = \overline{Q}_a$, a cada descida de *clock* (Figura 2.39 (a)). Inicialmente, é aplicado um nível 0 à entrada CLEAR para que todos os *flip-flops* fiquem com a saída igual a 0. A cada descida do pulso de *clock*, o primeiro *flip-flop* muda de estado, e essa troca é aplicada à entrada do segundo *flip-flop*, o qual troca de estado a cada descida da saída Q_0 e assim por diante.

O contador de década (Figura 2.39 (b)) também funciona de modo assíncrono. Esse tipo de circuito efetua a contagem em números binários de 0 a 9, ou seja, de 0000 até 1001. Para construir esse circuito, utilizamos o contador de pulsos, interligando as entradas CLEAR dos *flip-flops*.

Para o contador contar somente até 9, quando chegar no 1010, ou seja, 10 em decimal, devemos jogar um nível 0 na entrada CLEAR do *flip-flop*. Para isso, devemos ligar as saídas Q_3 e Q_1 em uma porta NE de duas entradas (CAPUANO, 2014).

Figura 2.39 | Contador de pulsos assíncrono (a) e contador de décadas (b)



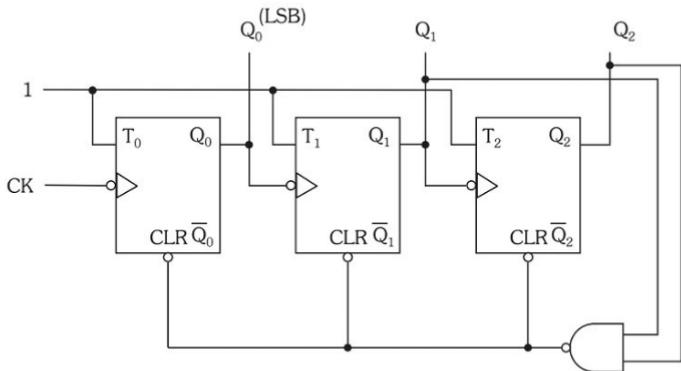
Fonte: Capuano (2014, p. 97-98).

Exemplificando

Vamos projetar o circuito de um contador que conte de 0 a 5, recomendando a contagem quando chegar no 6 e, assim, colocando em 0 todos os *flip-flops*.

No 6, 110 em binário, temos as saídas $Q_2 = 1$, $Q_1 = 1$ e $Q_0 = 0$, nesse caso devemos colocar 0 nas entradas CLEAR que estão interligadas, levando, dessa forma, o contador a 0. Para isso ocorrer, devemos colocar na entrada da porta NE, Q_2 e Q_1 .

Figura 2.40 | Contador



Fonte: Capuano (2014, p. 99).

Nos contadores síncronos, o *clock* é o mesmo em todos os *flip-flops*, ou seja, as entradas do *clock* estão curto-circuitadas. Para estudar os contadores síncronos, devemos escrever a tabela-verdade e verificar as entradas J e K dos vários *flip-flops*, para que eles assumam o estado seguinte. A partir da tabela do JK, construiremos a tabela dos contadores síncronos (Tabela 2.16) (CAPUANO, 2014).

Tabela 2.16 | Tabela de projeto dos contadores síncronos

Q_a	Q_f	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Fonte: elaborado pela autora.

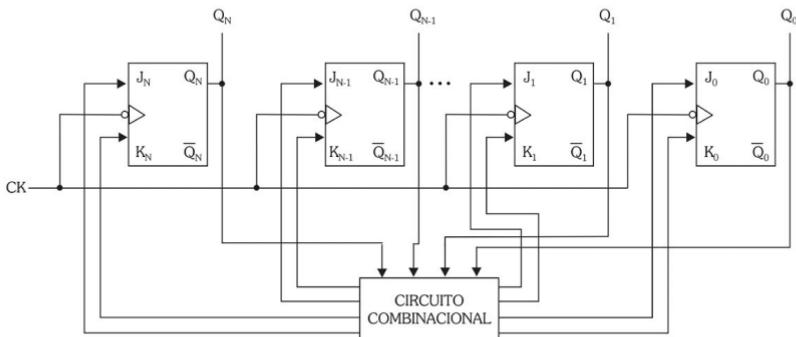
Vamos então analisar os quatro casos que temos:

- Primeiro caso: Q_a é 0 e o estado a ser assumido também é 0 ($Q_f = 0$). Nesse caso, podemos manter o estado do *flip-flop*, fazendo J = 0 e K = 0 ou podemos fixar 0, fazendo J = 0 e K = 1. Assim, J obrigatoriamente deve ser 0 e a entrada K pode ser 0 ou 1, condição de *don't care*.
- Segundo caso: Q_a é 0 e Q_f é 1. Assim, J obrigatoriamente deve ser 1 e a entrada K pode ser 0 ou 1, condição de *don't care*.

- Terceiro caso: Q_a é 1 e Q_f é 0. Nesse caso, K obrigatoriamente deve ser igual a 1 e a entrada J pode ser 0 ou 1, condição de *don't care*.
- Quarto caso: Q_a é 1 e Q_f é 1. Nesse caso, K obrigatoriamente deve ser igual a 0 e a entrada J pode ser 0 ou 1, condição de *don't care* (CAPUANO, 2014).

Após analisar o resultado da tabela-verdade das entradas JK dos *flip-flops* para a sequência desejada, faremos as simplificações e montaremos o circuito combinacional em função das saídas dos *flip-flops* (Figura 2.41).

Figura 2.41 | Contador síncrono genérico

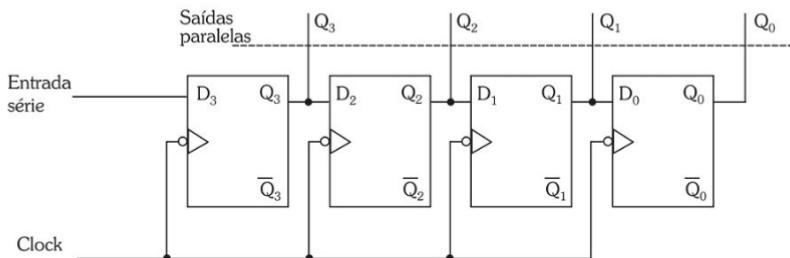


Fonte: Capuano (2014, p. 100).

O *flip-flop* é extremamente útil e importante, porém só pode armazenar um bit na saída Q . Quando for necessário armazenar mais de um bit, usamos o Registrador de Deslocamento (*Shift Register*). Um registrador de deslocamento é constituído de *flip-flops* tipo D que são ligados de modo que as saídas de cada bloco sejam aplicadas às entradas do *flip-flop* seguinte.

Uma das aplicações do registrador de deslocamento é a conversão de uma informação em série para outra em paralelo, ou seja, funciona como conversor série-paralelo. Para uma informação de 4 bits, temos uma configuração básica (Figura 2.42).

Figura 2.42 | Registrador de deslocamento



Fonte: Capuano (2014, p. 110).

Os sistemas sequenciais são utilizados em projetos para diversas finalidades, como os contadores que executam sequências numéricas a partir de um pulso de *clock*. Os sistemas sequenciais, que projetamos para aplicações específicas, também são chamados de máquinas de estado ou de máquinas de estados finitos, pois teremos um número finito de estados possíveis de saída. O termo máquina de estado se refere a um circuito que sequencia um conjunto de estados predeterminados e controlados por um *clock* e por outros sinais de entrada. Há dois modelos normalmente utilizados para a implementação de máquinas de estados: os modelos de Moore e de Mealy.

Assimile

Relembrando: no modelo de Moore, as saídas dependem do estado atual registrado internamente no sistema e as mudanças das saídas só acontecem no próximo *clock*. No modelo de Mealy, as saídas vão depender das entradas e do estado atual registrado internamente no sistema, ou seja, quando acontecerem mudanças nas entradas, as saídas também mudarão.

As máquinas de estado também podem ser classificadas em síncronas ou assíncronas de acordo com as características de aplicação. O modelo de Moore é síncrono e, no modelo de Mealy, as saídas podem atuar assincronamente.

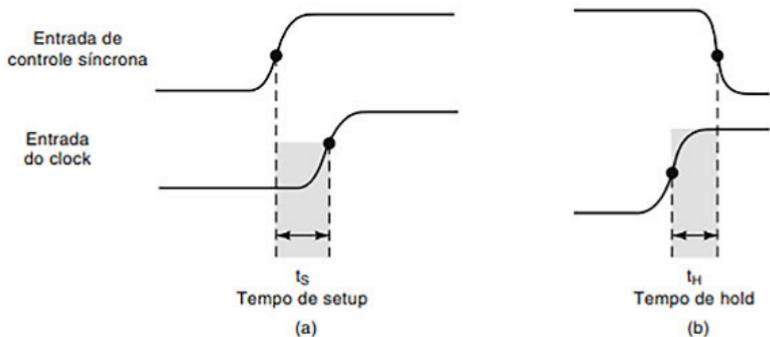
Assimile

Para efetuar os projetos de sistemas sequenciais, devemos determinar o diagrama de estados, levantar a tabela-verdade da sequência de estados proposta, colocar na tabela os estados a serem assumidos nas entradas dos *flip-flops*, simplificar os casos de saída através dos mapas de Karnaugh e elaborar o circuito com as expressões simplificadas.

Alguns parâmetros de temporização (especificados pelos próprios fabricantes) são importantes e devem ser considerados antes de usarmos um *flip-flop*.

O tempo de *setup* (preparação), t_s , e o tempo de *hold* (manutenção), t_h , mostrados na Figura 2.43, são parâmetros importantes de temporização que devem ser observados para que um *flip-flop* com *clock* responda de maneira correta às entradas de controle, ao ocorrer uma transição ativa na entrada *clock*. Esses tempos são medidos entre os instantes em que as transições estão em 50% e são importantes em sistemas síncronos, pois há situações em que as entradas de controle síncronas de um *flip-flop* mudam de estado aproximadamente ao mesmo tempo que a entrada *clock* (TOCCI; WIDMER; MOSS, 2011).

Figura 2.43 | Entradas de controle têm de ser mantidas estáveis por um tempo t_s antes da transição ativa do *clock* (a) e por um tempo t_h após a transição ativa do *clock* (b)



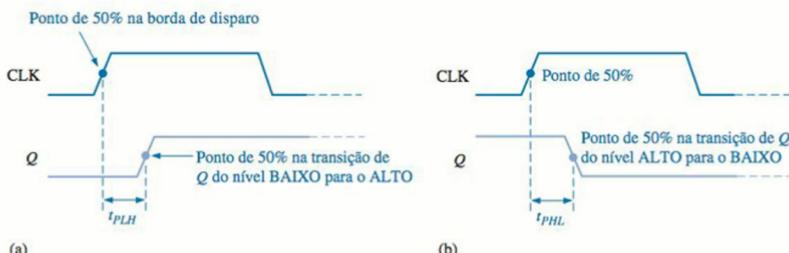
Fonte: Tocci, Widmer e Moss (2011, p.187).

O tempo de *setup* é o intervalo que antecede a transição ativa do sinal de *clock*, durante o qual a entrada de controle tem de ser mantida no nível adequado. O tempo de *hold* é o intervalo após a transição ativa do *clock*, durante o qual a entrada de controle síncrona tem de ser mantida no nível adequado. Para garantir que um *flip-flop* com *clock* responda de forma correta quando ocorrer a transição ativa do *clock*, as entradas de controle têm de estar estáveis por pelo menos um intervalo de tempo igual a $t_{s(\min)}$ antes da transição do *clock* e por pelo menos um intervalo de tempo igual a $t_{h(\min)}$ após a transição do *clock*. Os tempos de *setup* geralmente estão na faixa de 5 a 50 ns, e os tempos de *hold* estão na faixa de 0 a 10 ns (TOCCI; WIDMER; MOSS, 2011).

Toda vez que um sinal muda de estado na saída dos *flip-flops* há um atraso de tempo do instante em que o sinal é aplicado até o instante em que a saída

comuta de estado. Os atrasos de propagação (Figura 2.44) são medidos entre os pontos de 50% da amplitude das formas de onda de entrada e de saída.

Figura 2.44 | Atrasos de propagação nos *flip-flops*: atraso em transição de BAIXO para ALTO (a) e atraso em transição de ALTO para BAIXO (b)

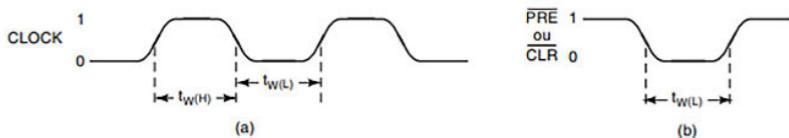


Fonte: Floyd (2007, p. 406).

As entradas assíncronas, PRESET e CLEAR também possuem os mesmos tipos de atrasos. Os fabricantes normalmente especificam os atrasos de propagação em resposta a todas as entradas. Os CIs mais novos de *flip-flops*, por exemplo, têm atrasos de propagação que vão desde alguns nanosegundos até valores em torno de 100 ns. Os valores de t_{PLH} e t_{PHL} normalmente não são os mesmos: eles aumentam proporcionalmente em relação ao número de cargas acionadas pela saída Q adiante.

A maior frequência que pode ser aplicada na entrada *clock* de um FF, mantendo um disparo confiável, varia de um *flip-flop* para outro, mesmo que eles tenham o mesmo número. Os fabricantes também especificam o tempo mínimo de duração que o sinal *clock* deve ficar no nível baixo antes de passar para o alto, e o tempo mínimo que o sinal *clock* deve ficar no nível alto antes de voltar para o baixo (Figura 2.45(a)). Caso esses tempos sejam desconsiderados, é possível que se resulte em disparos não confiáveis (TOCCI; WIDMER; MOSS, 2011).

Figura 2.45 | Tempos de duração do *clock* em nível baixo e em nível alto (a) e largura do pulso assíncrono (b)



Fonte: Tocci, Widmer e Moss (2011, p. 199).

Os fabricantes especificam o tempo mínimo de duração que as entradas PRESET ou CLEAR devem ser mantidas no estado ativo, a fim de setar ou resetar o *flip-flop* de modo confiável (Figura 2.45 (b)).

Para que se tenha um disparo confiável, os tempos de subida e de descida do *clock* devem ser bem pequenos. Se a transição demorar para ir de um nível para outro, o *flip-flop* pode não disparar ou disparar de modo instável. Por exemplo, o tempo de transição para dispositivos TTL normalmente é ≤ 50 ns e ≤ 200 ns para CMOS.

Alguns exemplos de CIs comerciais de *flip-flops* e de parâmetros de temporização são (Tabela 2.17): 7474 Duplo *flip-flop* D disparado por borda (TTL padrão); 74LS112 Duplo *flip-flop* J-K disparado por borda (TTL Schottky de baixa potência); 74C74 Duplo *flip-flop* D disparado por borda (CMOS de porta metálica) e 74HC112 Duplo *flip-flop* J-K disparado por borda (CMOS de alta velocidade) (TOCCI; WIDMER; MOSS, 2011).

Tabela 2.17 | Parâmetros de temporização de *flip-flop* (em nanosegundos)

	TTL 7474	TTL 74LS112	CMOS 74C74	CMOS 74HC112
t_s	20ns	20ns	60ns	25ns
t_h	5	0	0	0
t_{PHL} de CLK para Q	40	24	200	31
t_{PLH} de CLK para Q	25	16	200	31
t_{PHL} de \overline{CLR} para Q	40	24	225	41
t_{PLH} de \overline{PRE} para Q	25	16	225	41
$t_{W(L)}$ CLK tempo no nível BAIXO	37	15	100	25
$t_{W(H)}$ CLK tempo no nível ALTO	30	20	100	25
$t_{W(L)}$ em \overline{PRE} ou \overline{CLR}	30	15	60	25
f_{MAX} em MHz	15	30	5	20

Fonte: adaptada de Tocci, Widmer e Moss (2011, p. 200).

O temporizador 555 é um dispositivo, na forma de CI, bem versátil, robusto e amplamente utilizado já que pode ser configurado de duas maneiras diferentes: multivibrador monoestável ou um multivibrador astável

(oscilador). Um multivibrador astável não tem estado estável oscilando entre dois estados estáveis sem qualquer disparo externo (FLOYD, 2007). Esse CI é utilizado em muitas aplicações, como temporizadores (*timer*), geradores de onda (pulso) e osciladores. O 555 já foi utilizado em brinquedos, computadores, alarmes e até em naves espaciais.

A maioria dos sistemas digitais é constituída de circuitos combinacionais e de elementos de memória.

Nesta seção, vimos circuitos combinacionais que recebem sinais lógicos tanto das entradas externas quanto das saídas dos elementos de memória e cujo principal elemento de memória é o *flip-flop*, que nada mais é do que um conjunto de portas lógicas. Além disso, vimos vários tipos de *flip-flops* e suas aplicações, como contadores e registradores. Também vimos a importância dos parâmetros de temporização e o que é uma máquina de estado.

Espero que você tenha gostado desta seção! Bons estudos!

Sem medo de errar

Lembra-se da situação-problema apresentada no início da seção? Vamos solucioná-la agora!

Para um contador que conte até cinco, precisaremos de 3 bits, pois $2^3 = 8$ e de 3 *flip-flops*, um para cada saída. Comecemos pela montagem da tabela-verdade.

Tabela 2.18 | Tabela-verdade do contador para a câmera do *Pet Dog*

Clock de pulso	Q_2	Q_1	Q_0	CLEAR
Primeiro	0	0	0	1
Segundo	0	0	1	1
Terceiro	0	1	0	1
Quarto	0	1	1	1
Quinto	1	0	0	1
Sexto	1	0	1	1
Sétimo	1	1	0	0

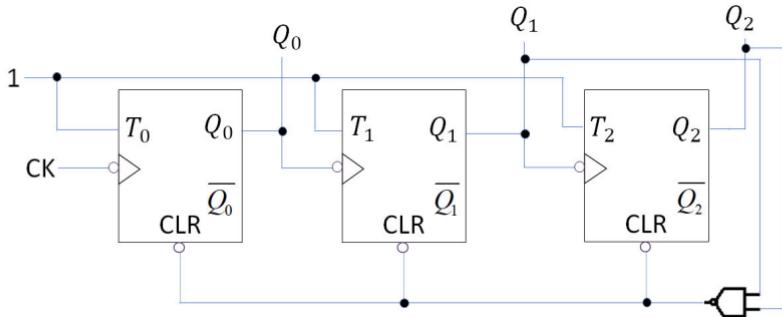
Fonte: elaborada pela autora.

Após a quinta descida de *clock*, o contador assume o estado $Q_0 = 1$, $Q_1 = 1$, $Q_2 = 0$, só que, neste instante, a entrada CLEAR vai para 0, zerando o contador,

ou seja, fazendo com que assuma o estado 0 (000) e reinicie a contagem. Portanto o CLEAR será: $\overline{Q_2} \overline{Q_1}$.

Para implementar esse circuito contador, vamos utilizar 3 *flip-flops* T e uma porta NAND, a qual será ligada nas saídas $\overline{Q_2}$ e $\overline{Q_1}$ do contador.

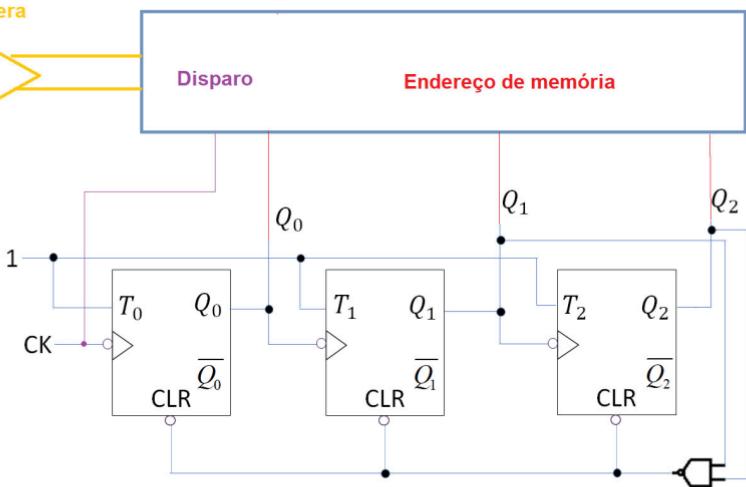
Figura 2.46 | Circuito contador para câmera do *Pet Dog*



Fonte: elaborada pela autora.

O contador é ligado na memória através das saídas Q_0 , Q_1 e Q_2 . O mais importante é ligar o disparo da câmera com o *clock* do contador, assim troca-se a posição em que a foto será salva.

Figura 2.47 | Circuito contador para câmera do *Pet Dog* com a ligação da memória e do *clock* camera



Fonte: elaborada pela autora.

Dessa forma, você entrega à diretoria o circuito do contador finalizado com a ligação da memória e o passo a passo de como você chegou ao resultado através da tabela-verdade do contador; da expressão que executa a tabela e do circuito do contador, utilizando o *flip-flop T*.

Avançando na prática

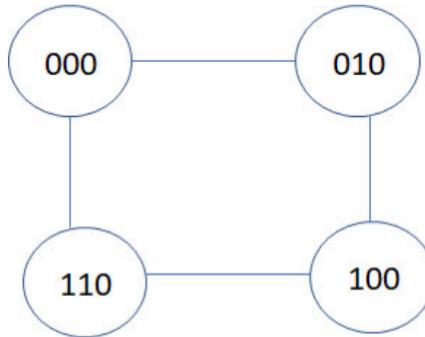
Contador em anel

Você foi contratado por uma empresa do ramo de turismo especificamente para implementar um contador nos hotéis dela, tendo em vista o aumento considerável na demanda. Trabalham na recepção duas moças. Uma delas só chama as senhas pares e a outra somente as senhas ímpares. Faça um contador em anel que conte somente os números pares de 0 a 6 e outro que conte de 1 a 5. Utilize *flip-flops JK*.

Resolução da situação-problema

A representação do contador em anel para o número par fica do seguinte modo:

Figura 2.48 | Contador em anel de números pares até 6



Fonte: elaborada pela autora.

Agora, vamos montar a tabela-verdade e, para isso, usaremos a seguinte tabela:

Tabela 2.19 | Tabela de projeto dos contadores síncronos

Q_a	Q_f	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Fonte: elaborada pela autora.

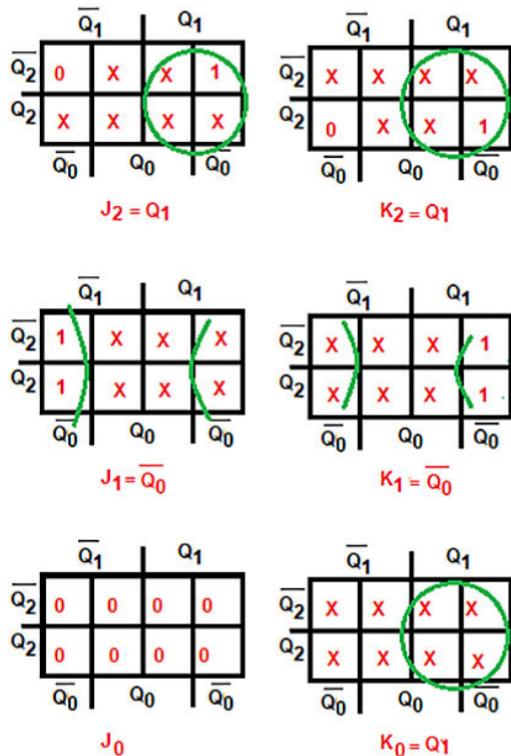
Tabela 2.20 | Tabela-verdade para o contador par

Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	X	1	X	0	X
0	1	0	1	X	X	1	0	X
1	0	0	X	0	1	X	0	X
1	1	0	X	1	X	1	0	X

Fonte: elaborada pela autora.

O próximo passo será montar o mapa de Karnaugh para cada uma das entradas JK e garantir a simplificação.

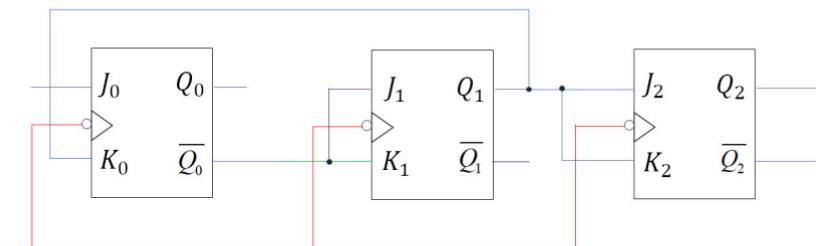
Figura 2.49 | Mapas de Karnaugh para cada entrada do *flip-flop*



Fonte: elaborada pela autora.

O circuito finalizado:

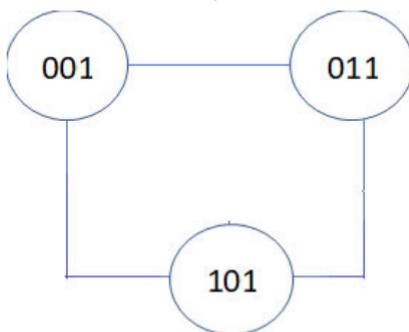
Figura 2.50 | Contador para números pares até seis



Fonte: elaborada pela autora.

O de número ímpar:

Figura 2.51 | Contador em anel de números ímpares até 5



Fonte: elaborada pela autora.

Agora, vamos montar a tabela-verdade. Para isso, usaremos a seguinte tabela:

Tabela 2.21 | Tabela de projeto dos contadores síncronos

Q_a	Q_f	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Fonte: elaborada pela autora.

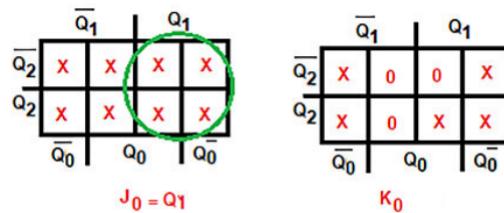
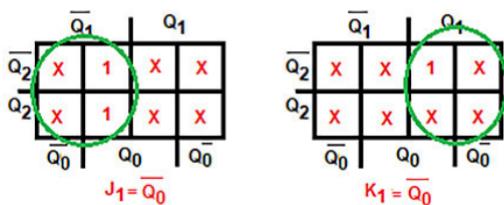
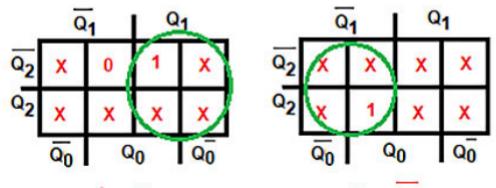
Tabela 2.22 | Tabela-verdade para o contador ímpar

Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	1	0	X	1	X	X	0
0	1	1	1	X	X	1	X	0
1	0	1	X	1	1	X	X	0

Fonte: elaborada pela autora.

O próximo passo será montar o mapa de Karnaugh para cada uma das entradas JK.

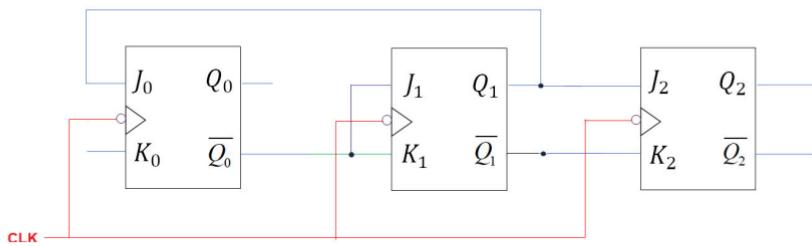
Figura 2.52 | Mapas de Karnaugh para cada entrada do *flip-flop*



Fonte: elaborada pela autora.

O circuito finalizado:

Figura 2.53 | Contador para números ímpares até 5



Fonte: elaborada pela autora.

Dessa forma, você entrega à empresa de turismo os dois contadores: pares e ímpares, mostrando passo a passo a resolução: tabelas-verdade, mapas de Karnaugh e os circuitos finalizados.

Faça valer a pena

1. Os sistemas sequenciais são utilizados em projetos para diversas finalidades, como os contadores que executam sequências numéricas a partir de um pulso de *clock*.

Considerando as informações apresentadas, analise as afirmativas a seguir:

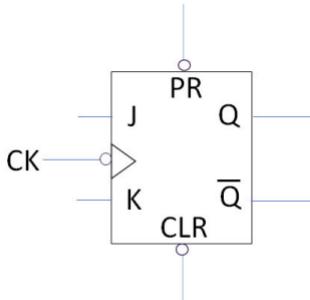
- I. O contador assíncrono tem um sinal de *clock* único aplicado externamente a todos os *flip-flops* ao mesmo tempo.
- II. O valor máximo de contagens que um contador que utiliza 5 *flip-flops* pode atingir é 32.
- III. Um *flip-flop* tipo D pode ser construído a partir de um *flip-flop* JK mestre-escravo com as entradas J e K curto-circuitadas.

Considerando o contexto apresentado, é correto o que se afirma em:

- a. I, apenas.
- b. II e III, apenas.
- c. I e III, apenas.
- d. II, apenas.
- e. III, apenas.

2. De forma simplificada, podemos representar o *flip-flop* através de um bloco com duas saídas, Q e \bar{Q} , de entradas para as variáveis, de uma entrada de controle (*clock*), de um CLEAR e de um RESET. A saída Q é a principal do bloco. A figura a seguir ilustra um *flip-flop* tipo JK mestre-escravo.

Figura | *Flip-flop* tipo JK mestre-escravo



Fonte: elaborada pela autora.

Assinale a alternativa que apresenta corretamente a sequência correta da tabela para o *flip-flop* JK mestre-escravo da figura.

Tabela | Tabela-verdade

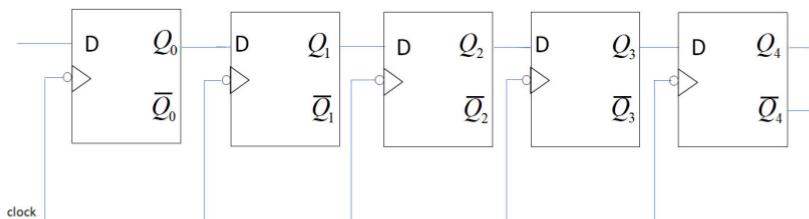
Clock	J	K	CLEAR	PRESET	Q_a	Q_f
1 \rightarrow 0	0	1	1	1	0	
1 \rightarrow 0	0	0	1	1	1	
1	1	0	0	1	0	
0	1	1	1	0	0	
1 \rightarrow 0	1	1	1	1	1	

Fonte: elaborada pela autora.

- a. 0; 1; 0 ;1; 0.
- b. 1; 0; 0; 0; 1.
- c. 0; 1; 1; 1; 0.
- d. 1; 1; 1; 1; 1.
- e. 0; 1; 0; 0; 0.

3. O *flip-flop* sozinho só pode armazenar um bit, mas, se necessitarmos guardar informações de mais de um bit, precisaremos utilizar um registrador de deslocamento (*shift register*).

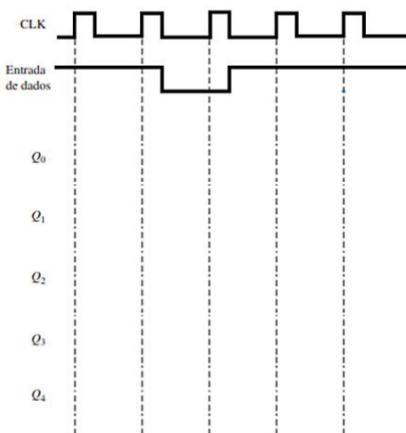
Figura | Exemplo de um registrador de deslocamento



Fonte: elaborada pela autora.

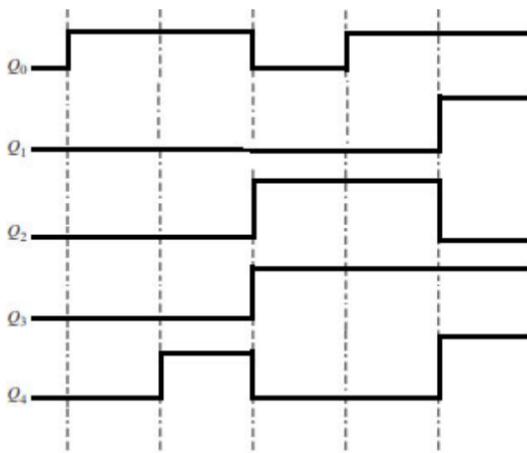
Assinale a alternativa que apresenta corretamente os estados do registrador de 5 bits, mostrado na figura para as formas de onda de entrada de dados especificada, e do clock. Inicialmente, o registrador está com todos os bits em nível 0.

Figura | Exemplo de entrada e saída do tipo flip-flop

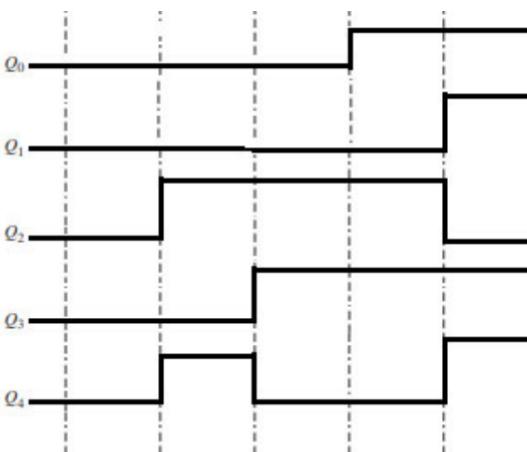


Fonte: elaborada pela autora.

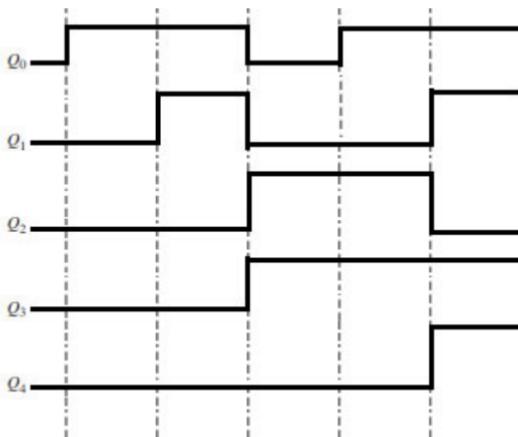
a.



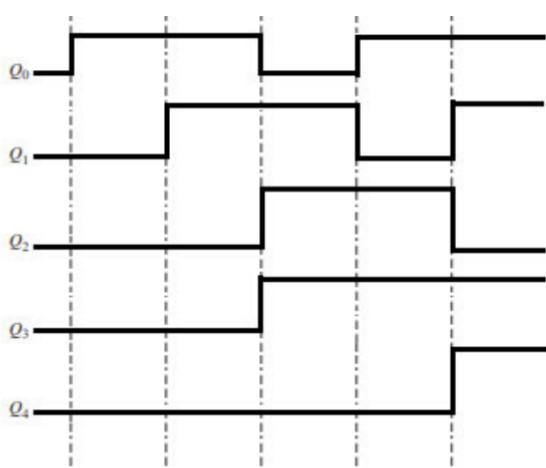
b.



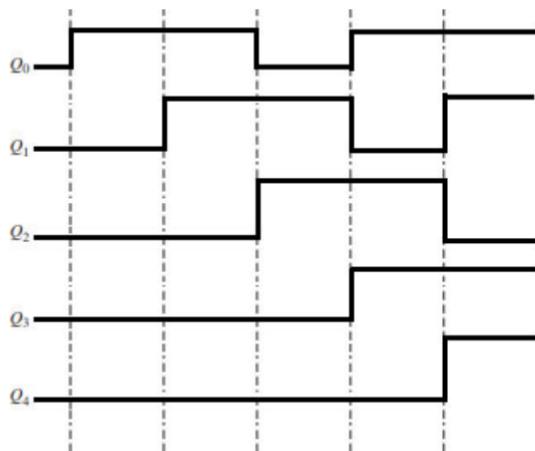
c.



d.



e.



Referências

- CAPUANO, F. G. **Sistemas digitais:** circuitos combinacionais e sequenciais. São Paulo: Érica, 2014.
- FLOYD, T. L. **Sistemas Digitais:** Fundamentos e Aplicações. 9. ed. Porto Alegre: Bookman, 2007.
- LATHI, B. P.; DING, Z. **Sistemas de Comunicações Analógicos e Digitais Modernos.** 4. ed. Rio de Janeiro: Editora LTC, 2012.
- RIBEIRO, G. C.; ANDRADE, H. G.; FONSECA, M. B. Controle de potência utilizando técnicas de modulação por ciclos inteiros. In: CONGRESSO DE INOVAÇÃO TECNOLÓGICA, 2., Pelotas, 2018. **Anais** [...]. Pelotas: UFPEL, 2018. Disponível em: https://cti.ufpel.edu.br/siepe/arquivos/2018/EN_00126.pdf. Acesso em: 20 mar. 2021.
- TOCCI, R.; WIDMER, N.; MOSS, G. **Sistemas Digitais:** Princípios e Aplicações. 11. ed. São Paulo: Pearson Prentice Hall, 2011.
- VAHID, F. **Sistemas digitais:** projeto, otimização e HDLs. Porto Alegre: Bookman, 2008.

Unidade 3

Giancarlo Michelino Gaeta Lopes

Arquitetura de microprocessadores e microcontroladores

Convite ao estudo

Prezado aluno,

De modo geral, a maioria dos sistemas embarcados da atualidade são digitais e possuem microprocessadores ou microcontroladores como elemento de controle. Assim, conhecer a arquitetura desses dispositivos digitais é fundamental para o profissional que trabalha com tecnologia.

Um sistema embarcado é definido como um dispositivo digital construído para o único propósito de sua aplicação. Por isso, um computador pessoal, que pode ser utilizado para diversas funções, não é um sistema embarcado. Por outro lado, diversos equipamentos eletroeletrônicos que utilizamos em nossas casas são assim categorizados. Exemplos são: forno micro-ondas, máquina de lavar, impressora, roteador e calculadora. Perceba que há uma vasta gama de equipamentos que possuem internamente um processamento digital que, dependendo de sua arquitetura de construção, irão demandar um microcontrolador ou um microprocessador.

Quando pensamos em equipamentos de controle simples, como uma máquina de lavar, nota-se a utilização de um microcontrolador como elemento principal. Esse dispositivo tem como principal característica possuir, dentro de um único chip, um processador em pequena escala e diversos periféricos. Agora, os equipamentos que realizam o processamento de uma grande quantidade de dados, como um roteador, podem exigir a utilização de um microprocessador dedicado, com poder de processamento maior que o encontrado em um microcontrolador.

Presumindo que, neste momento, você já possui algum conhecimento sobre o que são os microcontroladores e os microprocessadores e em que circunstâncias eles são utilizados, fica clara a importância de conhecer e compreender os principais conceitos sobre as arquiteturas desses dispositivos. Além disso, também é fundamental que você saiba projetar circuitos e sistemas que utilizem esses dois dispositivos. Dessa forma, esta unidade tem

por objetivo permitir que você tenha o conhecimento necessário para trabalhar com os microcontroladores e microprocessadores.

Na primeira seção, você irá estudar as diferenças entre um microcontrolador e um microprocessador, os tipos de arquiteturas desses dispositivos e quais elementos os compõem. A segunda seção explora a fundo a arquitetura RISC (*Reduced Instruction Set Computer*), apresentando os tipos de instruções, formatos e modos de endereçamento e componentes dessa arquitetura. E, por fim, a terceira seção trabalha os conceitos da arquitetura CISC (*Complex Instruction Set Computer*), apresentando a microprogramação, bem como exemplos desse tipo de máquina.

Bons estudos!

Seção 1

Introdução ao microprocessamento

Diálogo aberto

Prezado aluno,

A grande maioria dos equipamentos eletrônicos com que temos contato atualmente operam de forma digital e possuem, em seu interior, um dispositivo de processamento que pode ser um microcontrolador ou um microprocessador, dependendo da aplicação e da capacidade de processamento necessárias. Com isso, conhecer as arquiteturas que esses dispositivos possuem e como eles se diferenciam entre si é muito importante.

Quando tratamos de microcontroladores, é preciso ter em mente que existem diversos modelos no mercado com diferentes topologias e arquiteturas. Além disso, eles possuem diversos periféricos, os quais facilitam a solução dos mais diversos problemas. Somado a isso, a capacidade de programação o torna extremamente versátil e, por possuir um custo bastante acessível, torna-se uma opção imediata aos mais variados tipos de sistemas embarcados. Portanto, por um lado, é possível dizer que atualmente é mais simples e barato desenvolver uma solução utilizando um microcontrolador ao invés de utilizar diversos circuitos integrados de portas lógicas ou um circuito analógico. Por outro, os microprocessadores são utilizados em sistemas que necessitam de um maior poder de processamento, nos quais os periféricos não ficam integrados e devem ser adicionados externamente, gerando uma outra gama de aplicações.

Agora, nesta seção você será desafiado a elaborar um projeto de segurança para um sistema embarcado. Você já saberia responder alguns breves questionamentos? Para um projeto de sistemas embarcados, você utilizaria um microcontrolador ou microprocessador? Uma vez tendo respondido essa questão, quais seriam as principais especificações do componente escolhido? Quais parâmetros se sobressaem nesse tipo de projeto para que sua escolha seja feita? Veja que todas essas questões precisam ser respondidas e o faremos ao longo da seção.

Então, imagine que você trabalha em uma empresa de equipamentos eletrônicos no desenvolvimento de sistemas embarcados e recebeu uma nova demanda de seu superior. No momento, você deve especificar um microcontrolador para ser utilizado em um equipamento que será utilizado em um sistema de segurança e monitoramento. O sistema em questão possui diversos sensores detectores de presença e de abertura de portas, que devem

ser lidos de forma contínua. Caso algo indevido aconteça, o sistema deverá tocar um alarme e enviar uma SMS para o usuário.

A partir das informações passadas pelo sistema de segurança, você já tem uma noção de qual microcontrolador utilizar, porém um colega de trabalho o questionou sobre quais critérios você considera para selecionar um microcontrolador. Quais são as especificações que um microcontrolador pode possuir? Como determinar o microcontrolador a ser utilizado em um projeto? Nesse projeto do sistema de segurança, quais parâmetros se sobressaem para a seleção do microcontrolador?

Para deixá-lo apto a trabalhar com os microcontroladores e microprocessadores, esta seção irá trazer uma visão geral sobre os elementos que os constituem, as arquiteturas mais comuns dos sistemas computacionais e os elementos internos que compõem esses dispositivos.

E então, você está pronto para mais esse estudo? Seu desafio não será pequeno, portanto nunca deixe de se atualizar!

Não pode faltar

O surgimento dos microcontroladores foi dado a partir da evolução dos circuitos integrados (CIs). No início, os chips apresentavam pequenas quantidades de portas lógicas e o desenvolvimento dos circuitos eletrônicos era trabalho exclusivo de engenheiros, pois era uma tarefa que necessitava de muito conhecimento.

Contudo, o avanço na tecnologia permitiu que milhares de transistores pudessem ser instalados em um único chip, gerando a possibilidade de fabricação dos microprocessadores. Associados a essa nova invenção, surgiram os primeiros computadores, obtidos a partir da adição de periféricos externos, tais como memórias, linhas de entradas e saídas, temporizadores, entre outros, aos microprocessadores. O surgimento desses dispositivos programáveis provocou uma grande revolução na área eletrônica, uma vez que, antes da invenção dos microprocessadores, tornava-se muito caro e difícil alterar um circuito de controle de máquinas e sistemas. Todavia, os dispositivos programáveis podem ser facilmente alterados para atender às necessidades que possam aparecer.

Um novo aumento na densidade de elementos dentro dos chips acarretou o surgimento de dispositivos que continham processador e periféricos na mesma estrutura, dando origem aos microcontroladores. Portanto, define-se microcontrolador como um pequeno computador contido em um único encapsulamento, o qual comprehende um processador e periféricos, permitindo uma inteligência programável utilizada no controle de processos lógicos.

Atualmente os microcontroladores são usados nas mais diversas áreas, por exemplo: automobilística, automação, segurança, controle de tráfego,

entretenimento, entre outras. Também podemos encontrar diversos exemplos de equipamentos com microcontroladores no nosso dia a dia, como: eletrodomésticos, alarmes, brinquedos e outros.

Para garantir que não haja nenhuma confusão entre microprocessadores e microcontroladores, vamos mostrar que eles diferem entre si em muitos aspectos. O Quadro 3.1 apresenta as principais características de cada dispositivo, evidenciando a diferença entre eles.

Quadro 3.1 | Comparação entre microprocessadores e microcontroladores

Características	Microprocessadores	Microcontroladores
Periféricos	Necessita de periféricos externos.	Memória e periféricos integrados no chip.
Barramento	Largura elevada do barramento de endereços.	Barramento de endereçamento estreito só permitindo o gerenciamento de pequena quantidade de memória.
Gerenciamento de memória	Unidade de gerenciamento de memória integrada, possibilitando vários formatos de dados.	Tratamento de poucos formatos de dados (tipicamente 8, 16 e 32 bits).
Processamento	Coprocessador matemático e unidade lógica aritmética com suporte a operações numéricas complexas, tais como multiplicação em ponto flutuante.	Ausência de coprocessador e com unidade lógica aritmética limitada.
Endereçamento	Sofisticados modos de endereçamento para eficiente suporte às funções de linguagem em alto nível.	Limitados modos de endereçamento (tratamento ineficiente de funções de linguagem de alto nível).
Custo	Alto custo.	Baixo custo.
Consumo	Alto consumo de energia.	Baixo consumo de energia.
Ocupação no silício	Ocupação de grande área de silício.	Pequena área de silício.

Fonte: elaborado pelo autor.

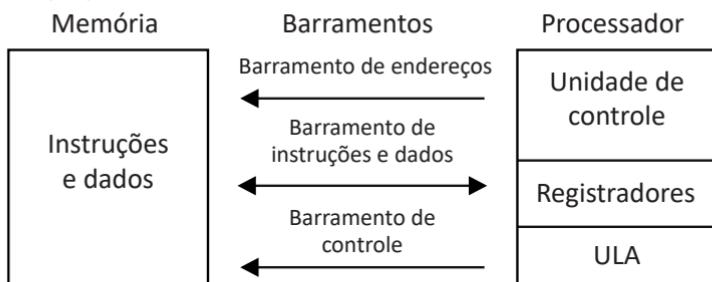
A principal diferença está relacionada a suas funcionalidades. Os microprocessadores são utilizados em aplicações que demandam cálculos numéricos complexos e em alta velocidade. Além do mais, para que eles possam ser utilizados, é necessária a adição de outros componentes, como memórias; eles também não são capazes de se comunicar com o ambiente externo por si só. Já os microcontroladores são utilizados de forma dedicada, em sistemas embarcados, em que a velocidade de processamento não é tão elevada. Para seu uso, não são necessários componentes externos, visto que todos os circuitos para sua aplicação estão dentro de um único chip.

É por meio da arquitetura de um sistema digital que se tem a definição de quem são e de como estão interligadas as partes que o compõem. Conhecê-la, auxilia no uso correto das instruções, na escolha da linguagem de programação e principalmente no uso da memória. As duas arquiteturas mais comuns para sistemas computacionais digitais são: Von Neumann e Harvard.

A Figura 3.1 ilustra a arquitetura proposta por **Von Neumann**. Sua principal característica é a utilização de um único sistema de memória, no qual são armazenados os dados (variáveis) e o programa a ser executado (software). Ela é composta por: (i) uma memória; (ii) uma unidade lógica aritmética (ULA); (iii) uma unidade central de processamento (CPU, do inglês *Central Processing Unit*), composta por registradores, e (iv) uma unidade de controle (UC), com função de buscar um programa na memória, instrução por instrução, e executá-lo de acordo com os dados de entrada.

Cada um dos elementos apresentados é realizado a partir de componentes físicos independentes, cuja implementação tem variado ao longo do tempo com o avanço da tecnologia, tendo início com o uso de relés e válvulas e, posteriormente, sendo utilizados circuitos integrados fabricados com semicondutores.

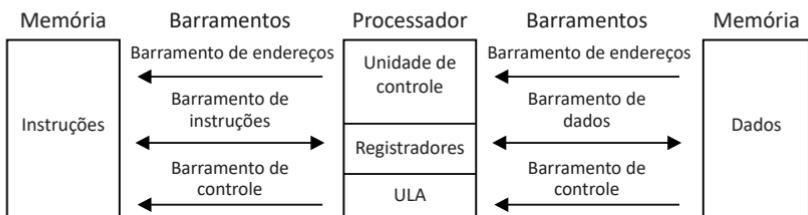
Figura 3.1 | Arquitetura Von Neumann



Fonte: elaborada pelo autor.

Já as máquinas que possuem arquitetura **Harvard** baseiam-se em um conceito mais recente que o de Von Neumann, possuindo duas unidades de memórias independentes, conforme ilustra a Figura 3.2. Portanto, as máquinas Harvard apresentam dois conjuntos de barramentos, um utilizado para acesso aos dados e outro para acesso às instruções de programa. Tipicamente o barramento de dados é de 8 bits, já o barramento de instruções pode variar entre 12, 14 e 16 bits. A diferença entre o número de bits dos barramentos decorre da utilização de memórias diferentes para armazenamento de instruções (memória ROM) e dados (memória RAM).

Figura 3.2 | Arquitetura Harvard



Fonte: elaborada pelo autor.

A principal vantagem dessa arquitetura é dada pela dupla ligação às memórias de dados e de programa, permitindo que o processador leia uma instrução ao mesmo tempo em que executa um acesso à memória de dados (*pipeline*).

A vantagem da arquitetura Von Neumann em relação à arquitetura Harvard, é que, por compartilhar o mesmo barramento de memórias e dados, os programas podem ser atualizados de acordo com as características de entrada do sistema. Contudo, a velocidade de processamento é seriamente reduzida em sistemas com muitos dados, pois o processamento é feito de forma totalmente sequencial. Já a arquitetura Harvard apresenta mais vantagens por possuir barramentos distintos. Conforme apresentado anteriormente, essa arquitetura permite uma velocidade maior de processamento, pois é capaz de executar dados em *pipeline*, ou seja, utiliza os barramentos simultaneamente. Também é possível apresentar tipos e tamanhos de memórias diferentes para o armazenamento do programa e dos dados.

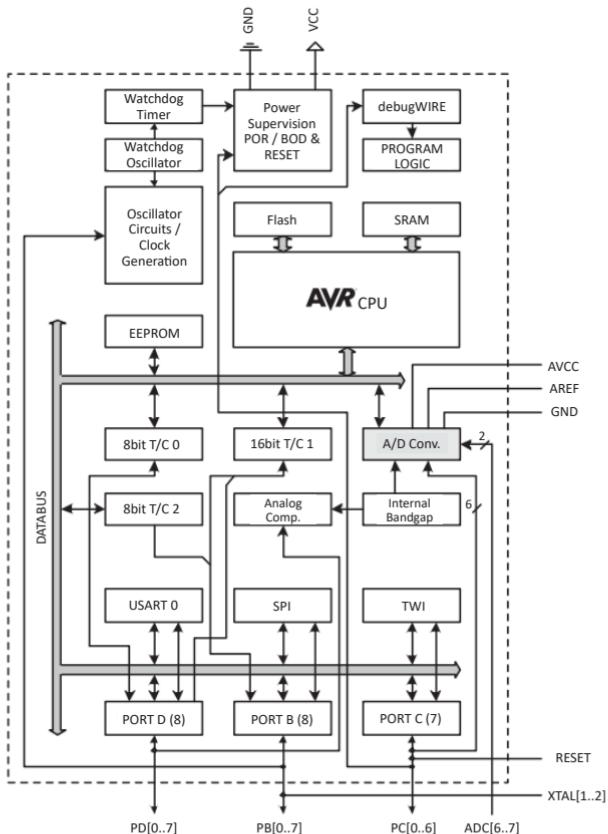
Nos dias de hoje, a arquitetura Von Neumann é mais utilizada em computadores, enquanto a arquitetura Harvard é mais presente nos microcontroladores (em sistemas embarcados) e DSPs (do inglês *Digital Signal Processor*, que são microprocessadores especializados em processamento digital de sinais) (MACHADO; MAIA, 2013).

Conhecendo as diferenças entre um microcontrolador e um microprocessador, bem como as arquiteturas Harvard e Von Neumann, nós estamos aptos a seguir os nossos estudos. Assim, a partir de agora, será trabalhada a estrutura de hardware de um microcontrolador. Porém, alguns conceitos devem ser definidos para que não haja confusão. Um sistema processado é formado pelo conjunto de hardware (componentes físicos do sistema) e software (conjunto de instruções executado pelo processador, também chamado de programa). Ressalta-se que, nos casos em que o software é gravado na memória do processador, ele recebe o nome de firmware.

Para fazer o estudo do hardware e da arquitetura interna de um microcontrolador, nosso estudo irá se basear no dispositivo ATmega328, que é fabricado pela Microchip e que possui arquitetura interna AVR, baseada na arquitetura Havard. Esse microcontrolador será utilizado como base dos nossos estudos pois ele está presente na plataforma de desenvolvimento Arduino Uno, que é uma das mais populares atualmente.

Analizando o diagrama de blocos da estrutura interna do ATmega328 (Figura 3.3), verifica-se um conjunto de circuitos necessários para iniciar e desenvolver suas atividades, como: CPU, memórias, osciladores, entradas e saídas e periféricos. Assim, a seguir será tratado o papel de cada elemento no funcionamento do microcontrolador, exceto o conjunto de periféricos, que será tratado em outro momento no livro.

Figura 3.3 | Diagrama de blocos da estrutura interna do ATmega328

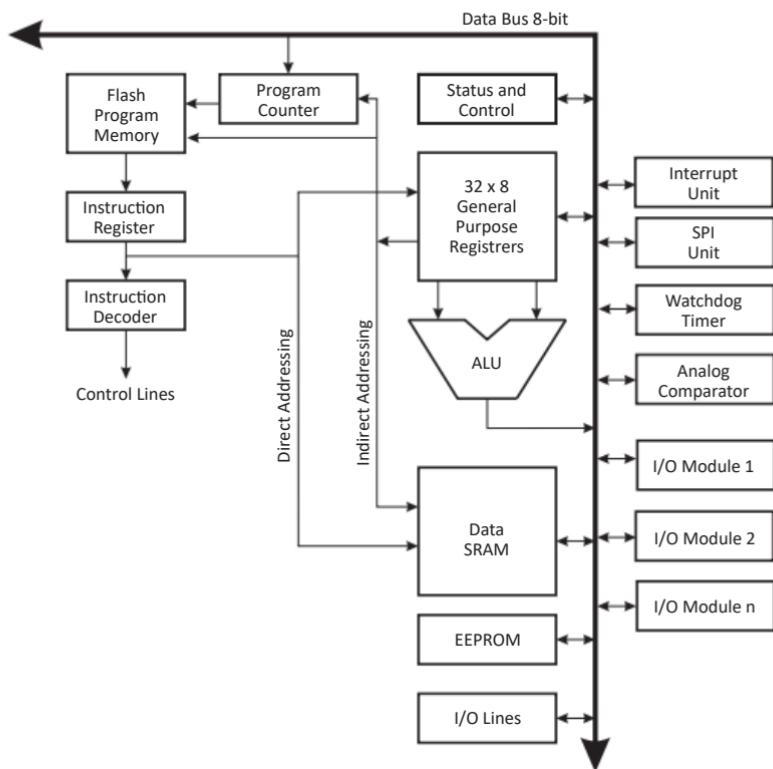


Fonte: Microchip (2018, p. 15).

A CPU, também chamada de núcleo (*core* em inglês), é a unidade interna mais importante do microcontrolador, pois ela é responsável por processar todos os dados do programa e executá-lo. Portanto, ela controla todas as demais partes do sistema através do envio e recebimento de dados e realiza as operações lógicas e aritméticas.

Na Figura 3.4, estão apresentados os elementos que constituem o núcleo do ATmega328. Ele é formado por várias subunidades, das quais se destacam: unidade lógico-aritmética (ULA ou ALU, do inglês *Arithmetic Logic Unit*), memórias (flash e SRAM), decodificador de instruções, contador de programa e registrador de instrução.

Figura 3.4 | Diagrama de blocos do núcleo do ATmega328



Fonte: Microchip (2018, p. 18).

Para que a execução de um programa ocorra adequadamente, ele deve ser composto por uma série de instruções em linguagem de máquina.

Dessa forma, as instruções devem estar armazenadas em posições sucessivas na memória principal, dado que a execução do programa é feita de forma sequencial, de modo que a próxima instrução a ser executada está contida uma posição à frente da instrução em execução. O ciclo de execução de uma instrução pelo processador passa pelas etapas de busca, decodificação e execução.

O contador de programa (PC – *Program Counter*) é o elemento responsável por armazenar a posição de memória da próxima instrução. Durante a execução do software, ocorre o processo de busca da instrução, momento em que o registrador de instruções (*Instruction Register*) deve ser preenchido pelas instruções contidas no endereço de memória apontado pelo PC. Quando uma nova instrução é iniciada, o PC é incrementado. Caso o sistema seja reiniciado, o contador de programa é carregado com o endereço da primeira instrução. Em casos de desvios no programa, o PC é alterado novamente, com o endereço do ponto do programa a ser executado.

Na execução de uma função, o fluxo do programa é desviado para buscar as instruções dessa determinada função e depois ele deve retornar ao ponto em que ocorreu a chamada. Para isso, o endereço da posição de memória em que o programa se encontra deve ser armazenado em algum registrador. O local onde esse valor é armazenado é chamado de Apontador de Pilha (*Stack-Pointer*). O nome pilha é devido ao seu funcionamento. Quando ocorrem várias chamadas de funções, os endereços de retornos vão sendo armazenados uns sobre os outros, como se fossem uma pilha de pratos. À medida que os endereços vão sendo coletados para o retorno do programa, a pilha vai se esvaziando.

Assim que a execução da instrução presente no registrador de instruções for iniciada, parte do registro vai para o decodificador de instruções e parte pode ir para os registradores ou para a unidade lógico-aritmética. Nos circuitos de decodificação, ocorre a interpretação da instrução, gerando sinais elétricos específicos a essa instrução. Esses sinais resultam na execução da instrução, que corresponde à aplicação de uma operação no conjunto de dados enviados aos registradores ou à ULA (chamado de operandos). Em outras palavras, o decodificador recebe a sequência de ações a serem executadas, e a ULA ou os registradores recebem os dados a serem processados.

A unidade lógico-aritmética é o circuito responsável por realizar os cálculos no processador. Suas funções englobam operações lógicas

(E, OU, OU-exclusivo, deslocamentos, rotações, complemento) e aritméticas (incremento, decremento, adição, subtração, multiplicação e divisão). Todavia, os processos de multiplicação e divisão são realizados a partir de uma sequência de somas e subtrações (OLIVEIRA; ZANETTI, 2015).

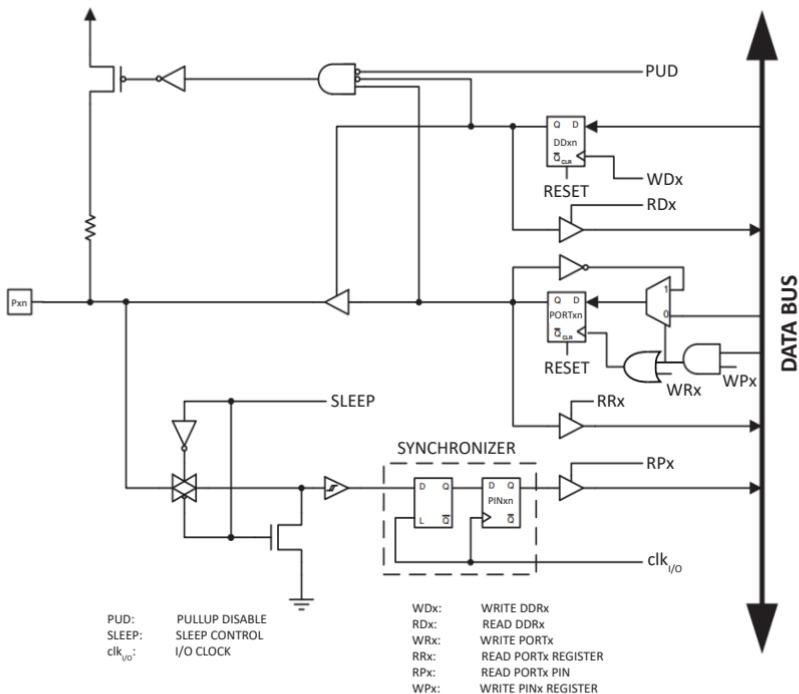
O registrador de *status* é responsável por indicar resultados notáveis das operações matemáticas através de *flags*. Assim, é possível saber se uma operação resultou em zero, se houve estouro da capacidade de armazenamento (*overflow*) ou se uma operação aritmética possui resultado positivo ou negativo.

Toda CPU conta com um sistema antitravamento, operado pelo temporizador *Watchdog*, que possui um oscilador RC interno para funcionar de forma independente ao sinal de *clock* do microcontrolador. Quando essa função está habilitada, o contador deve ser zerado em intervalos de tempo regulares, caso contrário, quando o contador estourar, o sistema é reiniciado. No microcontrolador ATmega328, o período do *Watchdog* típico é de 16 ms, podendo chegar a 8 s.

Os **dispositivos de entrada** e saída, comumente chamados de dispositivos I/O (do inglês, *Input/Output*) ou portas I/O, realizam a interface do controlador com o “mundo externo” por meio de um sistema computacional capaz de adquirir dados externos e também de acionar processos. Para isso, o microcontrolador recebe ou gera níveis lógicos de tensão.

Os dispositivos de entrada e saída ocupam praticamente todos os pinos do microcontrolador, exceto aqueles destinados a sinais de alimentação e tensões de referência. Cada elemento pode ser configurado, via software, como entrada ou saída, podendo, inclusive, alterar seu status no decorrer do programa. Para exercer a função de saída, ou seja, transmitir um dado que esteja presente no barramento de dados ao ambiente externo, são utilizados *latches*, que permitem carregar os dados, retendo-os mesmo que haja alterações em suas entradas. Já na configuração de entrada, a recepção de um sinal do ambiente externo para o barramento é feita por circuitos de *buffers* com saídas *tri-state*. Na Figura 3.5, é possível ver o circuito esquemático geral dos dispositivos I/O do ATmega328.

Figura 3.5 | Circuito esquemático geral das portas I/O



Fonte: Microchip (2018, p. 85).

Para armazenar as configurações das portas, são utilizados registradores. Assim, cada conjunto de portas apresenta, tipicamente, uma quantidade de entradas/saídas referente ao número de bits dos registradores.

Exemplificando

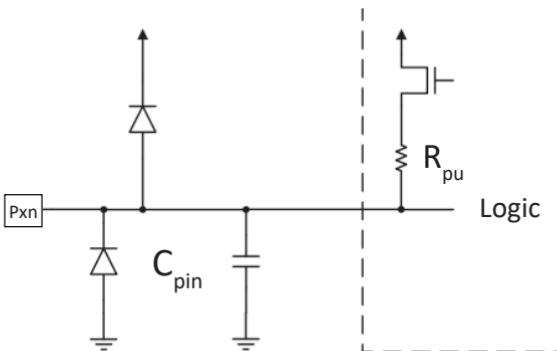
Para o ATmega328, que possui registradores de 8 bits, as portas são divididas em três conjuntos (Porta B, Porta C e Porta D), que apresentam até oito dispositivos I/O (P00, P01, P02, P03, P04, P05, P06 e P07). Dessa forma, cada bit do registrador determina a função de um pino correspondente. Caso o bit apresente nível lógico 1, o pino é configurado como entrada. Para nível lógico zero, sua função é de saída (MICROCHIP, 2018).

Uma importante especificação dos dispositivos I/O é a corrente máxima que consegue ser drenada em cada pino. Para a maioria dos microcontroladores atuais, a corrente máxima suportada é de aproximadamente 20mA. Usualmente, quanto mais pinos, menor a corrente máxima de cada um. Para o

ATmega328, a corrente máxima em cada pino é de 40mA, entretanto, a soma das correntes de todo o CI não pode ultrapassar 200mA (MONK, 2014).

Uma outra característica importante dos pinos de I/O é que eles podem ter resistências de *pull-up*, que os conecta à tensão positiva de alimentação. Tal ligação é bem útil para utilização de botões, interruptores, opto-acopladores, entre outros. Também existe um sistema de proteção contra sobre tensões ou subtensões, cuja funcionalidade é equivalente à utilização de diodos, conforme apresentado na Figura 3.6, que demonstra o circuito esquemático equivalente dos dispositivos de entrada/saída.

Figura 3.6 | Circuito esquemático equivalente dos dispositivos de entrada/saída



Fonte: Microchip (2018, p. 84).

Além da função de entrada e saída, a maioria dos pinos são multiplexados com funções alternativas, correspondentes a outros periféricos (Timers, USART, etc). Essas utilidades são específicas de cada pino e acabam definindo sua utilização.

A **memória** é o dispositivo responsável por armazenar dados. Para facilitar seu entendimento, podemos compará-la a um armário de arquivos com muitas gavetas. Suponha que estas são identificadas com uma etiqueta na parte frontal, para que seu conteúdo possa ser encontrado com facilidade. O mesmo ocorre com as memórias. A identificação de um espaço da memória, disponível para armazenamento de dados, é identificada pelo seu endereço. Assim, um conteúdo pode ser escrito ou lido pelo processador, dado seu endereçamento.

O microcontrolador abordado neste material, o ATmega328, apresenta arquitetura Harvard, consequentemente possui unidades de memória separadas para armazenamento de instruções e de dados. Isso possibilita que as memórias apresentem características diferentes uma da outra. Além disso, ele apresenta uma terceira memória não volátil, também utilizada para armazenamento de dados.

Na sequência, temos as características dos principais tipos de memórias.

- Memória apenas de leitura (*Read Only Memory – ROM*): é usada para armazenar permanentemente o programa, pois não permite uma nova escrita que possa sobrepor os dados.
- Memória *Flash*: é uma memória não volátil, apagável eletricamente e com tempos de acesso para leitura bem rápidos. Oferece uma grande flexibilidade nos projetos, pois é possível reprogramar um equipamento sem substituição de componentes.
- Memória de acesso aleatório (*Random Access Memory – RAM*): quando a alimentação do sistema é desligada, seus dados são apagados. É utilizada para armazenar dados temporários e resultados intermediários criados e utilizados durante a execução do software.
- Memória ROM eletricamente apagável e programável (*Electrically Erasable Programmable ROM – EEPROM*): por ser uma memória não volátil, ou seja, seus dados não são apagados com a interrupção da alimentação, ela é utilizada para armazenar dados que devem ser permanentemente guardados.

A memória de programa é definida pela unidade de memória responsável por armazenar o software. Portanto, é necessário que ela seja não volátil. Já a memória de dados é utilizada para armazenar variáveis e registradores utilizados no decorrer das instruções, por conseguinte são utilizadas memórias do tipo RAM.

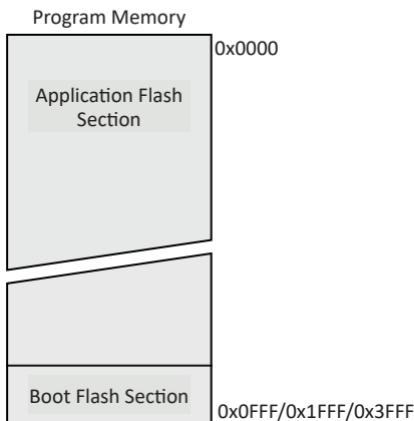
Exemplificando

O ATmega328 é composto por uma memória *Flash* de 32 kB para armazenamento do programa. Por isso, sempre é recomendado que o programador se preocupe com a qualidade e o tamanho do código, evitando estourar sua capacidade. Por apresentar instruções de 16 ou 32 bits, sua memória é organizada em 16k x 16. Isso significa que sua capacidade de armazenamento é de 16384 palavras de 16 bits cada (lembrando que 1k, em sistemas digitais, corresponde a 1024). Portanto, os endereços de memória apresentam valores de 0x0000 a 0x3FFF.

Outra característica importante é que essa memória é dividida em duas seções, conforme demonstra a Figura 3.7. A primeira, seção de aplicação (*Application Flash Section*), corresponde ao local onde o programa é armazenado. A segunda, seção de boot (*Boot Flash Section*), é utilizada para armazenar o programa *Boot Loader*, que permite a carga/descarga do código por qualquer meio ou protocolo de comunicação disponível. Graças a essa seção, a programação do Arduino se torna simples, utilizando somente um cabo USB no computador. Para outros tipos de

microcontroladores, a gravação do chip está associada à utilização de um gravador externo, que contém o *Boot Loader*.

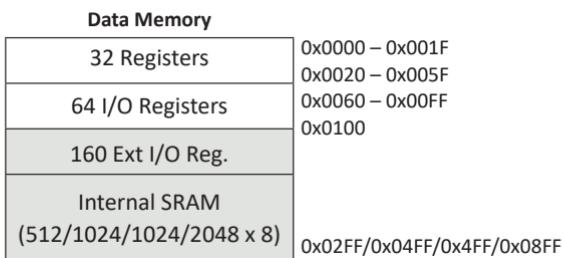
Figura 3.7 | Mapa da memória de programa do ATmega328



Fonte: Microchip (2018, p. 27).

Para o armazenamento de dados, o ATmega328 utiliza uma memória RAM de 2 kB, na qual são alocados os registros de propósito geral e dados dos dispositivos de entrada e saída. Além disso, uma quantidade da memória é utilizada durante a execução do código, armazenando resultados intermediários de operações lógicas e aritméticas e dados temporários. A organização estrutural da RAM é apresentada na Figura 3.8.

Figura 3.8 | Mapa da memória de dados do ATmega328



Fonte: Microchip (2018, p. 28).

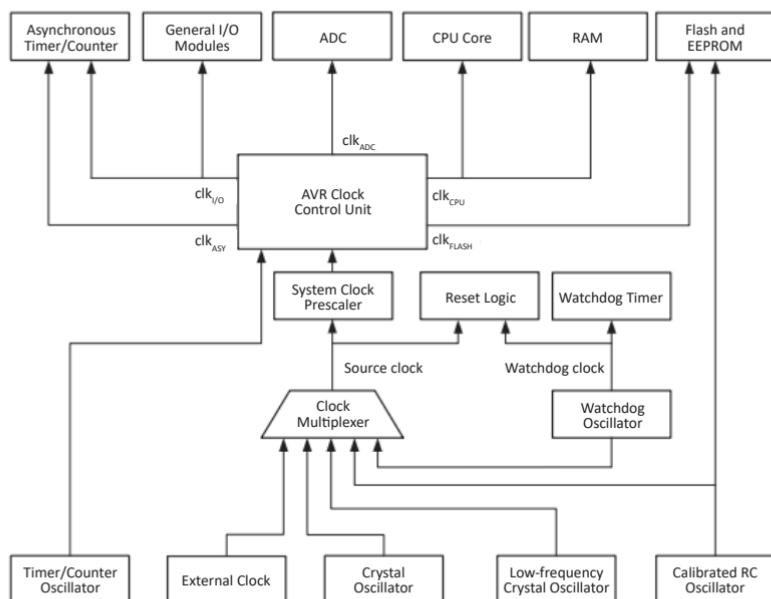
Os primeiros 32 endereços de memória armazenam os arquivos de registro de propósito geral e são utilizados para otimizar o desempenho na execução das instruções. Os 64 endereços seguintes salvam dados dos dispositivos de entrada e saída. Contudo, esse microcontrolador

apresenta mais unidades de periféricos do que os 64 endereços podem suportar. Por isso, foram estendidos mais 160 endereços para dispositivos I/O. Os endereços restantes atendem aos dados internos.

A terceira memória presente no ATmega328 é uma EEPROM, também utilizada para armazenamento de dados. Contudo, por ser uma memória não volátil, ela é utilizada para armazenar dados ocasionais de configuração do sistema ou variáveis que devem ser carregadas toda vez que o sistema for iniciado. Para esse microcontrolador, a EEPROM possui uma capacidade de armazenamento de 1 kB de dados, e permite até 100 mil ciclos de apagamento e escrita.

O microcontrolador realiza a execução de suas instruções por meio de uma base de tempo intitulada ciclos de máquina, na qual cada instrução possui um tempo de execução condizente com um múltiplo de um ciclo de máquina, obtido por meio de um sinal pulsante, comumente uma onda quadrada, na qual cada pulso corresponde a um ciclo. Esse sinal oscilante é chamado de sinal de relógio (*clock*) e é proveniente de um circuito oscilador. Atualmente, os microcontroladores apresentam diversas opções de utilização de circuitos osciladores. Na Figura 3.9, é demonstrada a distribuição do sinal de *clock* para as estruturas internas do ATmega328, que suporta sinais osciladores de até 20 MHz.

Figura 3.9 | Distribuição do *clock* para o ATmega328



Fonte: Microchip (2018, p. 36).

Apenas duas possibilidades de circuitos osciladores são provenientes da estrutura interna do ATmega328, que correspondem ao oscilador RC do *Watchdog* e ao oscilador RC calibrado. As demais opções necessitam da instalação de componentes externos para o fornecimento do sinal. Portanto, o sinal de *clock* externo pode ser gerado por: cristal oscilador cerâmico externo, cristal de baixa frequência externo e fonte de *clock* externo. Porém, a forma mais comum é o uso de cristais osciladores externos, que apresentam uma precisão maior e também permitem escolhas de frequências diferenciadas, enquanto os osciladores internos apresentam valores de frequências fixos.

Vale ressaltar que, conforme demonstra a Figura 3.9, as estruturas internas apresentam sinais de *clock* diferentes (clk_{ASY} , $\text{clk}_{\text{I/O}}$, clk_{ADC} , clk_{CPU} e $\text{clk}_{\text{FLASH}}$), derivados da fonte primária. Isso é feito para permitir um modo de baixo consumo de energia, ativando somente os sinais em uso. Outra característica importante é o circuito de Prescaler, usado para diminuir a frequência do sinal de *clock* primário a fim de, também, diminuir o consumo de energia pelo controlador.

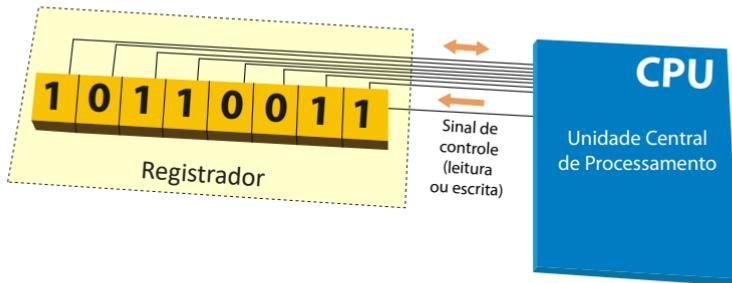
Refletá

Para que o microcontrolador execute uma instrução, são necessários quantos pulsos de *clock*? Isso depende da arquitetura do microcontrolador?

Outro elemento fundamental na estrutura de um microcontrolador são os registradores. Um **registraror** nada mais é do que uma célula de memória responsável por armazenar o estado de um ou mais bytes. No ATmega328, que é um microcontrolador de 8 bits, cada registraror é capaz de armazenar o estado de um byte, que são usados para guardar informações do programa, estado de entradas ou saídas, resultados de operações matemáticas, entre outros. Portanto, cada registraror pode apresentar um valor de 0 (0b00000000 ou 0x00) a 255 (0b11111111 ou 0xFF) e, para facilitar a visualização do status de cada bit, é preferível representá-lo em binário ou, para uma representação compacta, em hexadecimal.

Os registradores podem ser classificados em: registradores de propósito geral (*General Purpose Register* – GPR) e registradores de funções especiais (*Special Function Register* - SFR). Os GPR são utilizados para armazenamento de dados sobre os quais serão realizados algum tipo de operação. Eles não agem diretamente sobre circuitos eletrônicos, portanto, são utilizados novamente pela própria CPU para um novo cálculo ou processamento (SOUZA, 2010). A Figura 3.10 ilustra o funcionamento de um GPR se comunicando com a CPU. Na Figura 3.4 é possível observar um bloco indicando os registradores GPR ligados à ALU do ATmega328.

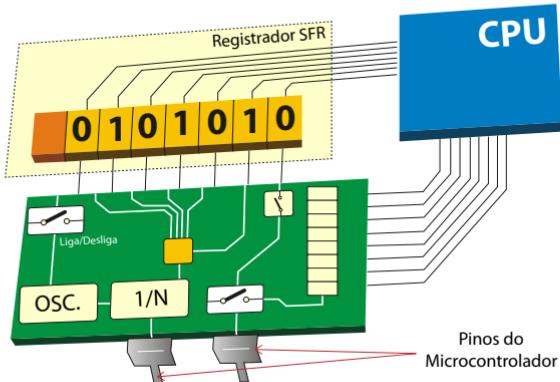
Figura 3.10 | CPU comunicando-se com um registrador de propósito geral



Fonte: adaptada de Verle (2010, [s.p.]).

Já os SFR possuem funções preestabelecidas pelo fabricante, uma vez que seus bits são conectados diretamente a circuitos internos do microcontrolador. Eles atuam diretamente no comando das operações de circuitos eletrônicos, como temporizadores, conversores A/D, dispositivos de entrada e saída e outros, guardando suas configurações e o estado atual da máquina (SOUZA, 2010). Cada controlador apresenta um número fixo de SFR de acordo com suas características. Um exemplo de registrador de função especial se comunicando com a CPU é apresentado na Figura 3.11.

Figura 3.11 | CPU comunicando-se com um registrador de função especial



Fonte: adaptada de Verle (2010, [s.p.]).

Os registradores de funções especiais (SFR) também apresentam uma subclassificação baseada nos tipos de circuitos eletrônicos que eles configuram, que são:

- SFR da CPU: responsáveis pelo controle e monitoramento dos processos da CPU. Apesar de apresentarem poucos registradores, a operação de todo o microcontrolador depende de seu conteúdo.
- SFR de periféricos: controlam a operação dos periféricos (temporizadores, conversores A/D, comunicação serial, etc.), na qual o periférico apresenta um ou mais registradores de controle.

Assimile

A programação de um microcontrolador pode ser feita a partir da escrita direta de seus registradores, porém também podem ser utilizadas bibliotecas que facilitam a programação. Assim, é importante saber que a plataforma Arduino se caracteriza por ter um ambiente de desenvolvimento simplificado, pois apresenta uma série de bibliotecas que evitam que o programador altere o conteúdo de um registrador com um comando direto. Por outro lado, essa facilidade de programação acarreta em um uso maior da memória, visto que operações que podem ser realizadas diretamente sobre o registrador são feitas através de funções armazenadas nessa memória.

A partir do que foi apresentado nesta seção, você pôde compreender a fundo todas as estruturas que compõem um microcontrolador. Além disso, você estudou as diferenças entre os microcontroladores e os microprocessadores.

Sem medo de errar

Você é um profissional que trabalha no desenvolvimento de sistemas embarcados e deve especificar um microcontrolador para ser utilizado em um equipamento que faz parte de um sistema de segurança e monitoramento. Analisando as características do projeto, você percebeu que são necessários diversos pinos de entrada e saída, bem como interfaces de comunicação para outros periféricos externos. Porém, existem outras considerações a serem feitas para a seleção do microcontrolador a partir das necessidades desse projeto.

A escolha de um microcontrolador para o desenvolvimento de um projeto não é uma tarefa simples. O grande número de modelos disponíveis no mercado pode confundir o projetista sobre qual microcontrolador utilizar, assim, antes de começar a projetar um dispositivo que contenha um microcontrolador, devem ser verificadas as necessidades do projeto, por exemplo:

- Quantidade de linhas de entrada/saída.
- Velocidade de processamento de dados.
- Consumo de energia.
- Quantidade de memórias.
- Quantidades e tipos de periféricos (comunicação serial, *timers*, memórias, outros).
- Interface de barramento de comunicação (CAN, I2C, SPI).
- Quantidade de canais e número de bits de conversores analógicos-digitais.
- Custo por unidade.
- Facilidade de upgrade.
- Ferramentas de desenvolvimento de software (*debugadores*, compiladores e suporte técnico).
- Tecnologia moderna para que o modelo adotado não desapareça do mercado em pouco tempo.

Estabelecida sua necessidade, você pode começar a verificar os modelos de microcontroladores dos fabricantes com os quais você já possui alguma familiaridade. Para iniciantes, os microcontroladores fabricados pela Microchip são os mais recomendados devido ao volume de documentação disponível, em especial dos dispositivos da família PIC e AVR. Além disso, pode-se optar pela utilização de uma placa de desenvolvimento para o protótipo inicial.

Para finalizar a resolução dessa situação problema, que tal consultar os sites de alguns fabricantes de microcontroladores? Assim, você poderá verificar a gama de modelos existentes e encontrar um modelo que possa ser utilizado para o projeto desse equipamento para sistema de segurança. As principais fabricantes de microcontroladores atuais são a STMicroelectronics, NXP e Texas Instruments, além da já citada Microchip.

Após a análise das características que um microcontrolador possui, foi possível resolver esta situação-problema, na qual foram apresentadas as principais especificações a serem consideradas para a determinação de um microcontrolador que será utilizado em um projeto.

Análise dos dispositivos de entrada e saída de um microcontrolador

Você possui uma empresa que presta serviços de desenvolvimento de sistemas embarcados e foi questionado por um cliente, que tem conhecimentos de eletrônica digital, sobre os circuitos internos que existem em um microcontrolador. O cliente já sabe que os dispositivos de entrada e saída ocupam praticamente todos os pinos do microcontrolador, exceto aqueles destinados a sinais de alimentação e tensões de referência. Além disso, ele sabe que cada elemento pode ser configurado via software como entrada ou saída, o que pode, inclusive, alterar seu status no decorrer do programa.

Para o projeto que está desenvolvendo a seu cliente, você está utilizando o microcontrolador ATmega328. Como ele possui alguma experiência com esse microcontrolador, pede para que você descreva os circuitos internos que são utilizados para os pinos do microcontrolador exercerem a função de entrada e de saída. Um outro questionamento dele é sobre quais elementos são utilizados para armazenar as configurações das portas. Como responder essas dúvidas do seu cliente?

Resolução da situação-problema

Em resposta, você explica que, para exercer a função de entrada, ou seja, a recepção de um sinal do ambiente externo para o barramento, são utilizados circuitos de *buffers* com saídas *tri-state*. Já para o funcionamento como saída, ou seja, para transmitir um dado que esteja presente no barramento de dados ao ambiente externo, são utilizados *latches* que permitem carregar os dados, retendo-os mesmo que haja alterações em suas entradas.

Para o armazenamento das configurações das portas, que vão dizer se ela irá funcionar como entrada ou como saída, são utilizados registradores. No caso, trata-se de registradores de funções específicas e existem diferentes registradores para a configuração dos pinos como entrada ou saída e para o armazenamento dos níveis lógicos dos pinos. Na maioria dos casos, também existe outro registrador que é utilizado para indicar a utilização ou não de resistores de *pull-up* ligados nos pinos.

Assim, apresentando essas características sobre os circuitos internos que a interface de entrada e saída do microcontrolador possui e explicando

a funcionalidade dos registradores, você conseguiu sanar as dúvidas de seu cliente.

Faça valer a pena

1. A arquitetura de um sistema digital define quem são e como estão interligadas as partes que o compõem. As duas arquiteturas mais comuns para sistemas computacionais digitais são: Von Neumann e Harvard.

Sobre as arquiteturas Von Neumann e Harvard, assinale a alternativa correta.

- a. A arquitetura Von Neumann utiliza três sistemas de memória, um para os dados, um para o programa e um de backup.
- b. A arquitetura Harvard apresenta apenas um barramento, por onde trafegam dados e instruções.
- c. Uma vantagem da arquitetura Von Neumann é permitir que o processador leia uma instrução enquanto executa um acesso à memória de dados.
- d. Uma vantagem da arquitetura Harvard é possuir duas memórias e consequentemente dois barramentos.
- e. As duas arquiteturas são muito semelhantes e ambas apresentam a mesma quantidade de memórias e barramentos.

2. A CPU, também chamada de núcleo (*core* em inglês), é a unidade interna mais importante do microcontrolador. Ela é formada por várias subunidades, entre as quais se destacam: unidade lógico-aritmética (ALU), memórias (*flash* e SRAM), decodificador de instruções, contador de programa e registro de instrução.

Sobre as subunidades da CPU, assinale a alternativa correta.

- a. O contador de programa é responsável por armazenar a posição de memória da instrução que está sendo executada pelo processador.
- b. O registro de instruções recebe as instruções contidas no endereço de memória apontado pela unidade lógico-aritmética após a execução de uma operação.
- c. A unidade lógico-aritmética é responsável por armazenar os dados lógicos e de operações aritméticas que a CPU irá realizar.
- d. O decodificador de instruções possui circuitos que interpretam a instrução recebida e geram sinais elétricos específicos a essa instrução.

- e. As memórias RAM são responsáveis por armazenar as instruções a serem executadas, retendo os dados mesmo em caso de desligamento do microcontrolador.

3. A base de tempo para o funcionamento de um microcontrolador é obtida através de um sinal pulsante, normalmente uma onda quadrada, na qual cada pulso corresponde a um ciclo, chamado de sinal de *clock*, que é proveniente de um circuito oscilador. Por meio de configuração, pode-se optar por utilizar um circuito oscilador presente em estrutura interna ou um montado com componentes externos ao microcontrolador.

Considerando o microcontrolador ATmega328 como referência, assinale a alternativa correta sobre os osciladores.

- a. Quando se deseja uma maior precisão na base de tempo, utiliza-se o oscilador RC do *Watchdog*.
- b. Os osciladores externos que podem ser utilizados são o oscilador RC do *Watchdog* e a fonte de *clock* externa.
- c. Os osciladores presentes na estrutura interna do ATmega328 são o oscilador a cristal de baixa frequência e o oscilador RC calibrado.
- d. Uma fonte de *clock* externa pode ser montada com o auxílio de um circuito gerador de *clock* externo.
- e. A utilização de um oscilador RC causa atraso na execução das instruções e travamento constante do microcontrolador.

Seção 2

Arquitetura RISC

Diálogo aberto

Prezado aluno,

É importante destacar que, ao se especificar um microcontrolador ou microprocessador, a arquitetura deste poderá ser dividida em duas grandes categorias a partir do conjunto de instruções que esses dispositivos utilizarão para funcionar. Assim, o dispositivo digital pode ser construído baseado na arquitetura RISC (*Reduced Instruction Set Computer*) ou CISC (*Complex Instruction Set Computer*).

A arquitetura RISC tem por característica possuir unidades de controle mais simples, que podem ser integradas em um maior nível, de modo que é muito utilizada em dispositivos móveis e em aplicações de tempo real, ou seja, sistemas normalmente ditos embarcados, como uma calculadora ou um controle remoto, que são muito utilizados em nosso dia a dia. Sabendo da importância dessa arquitetura, esta seção tem como objetivo estudar todas as características da arquitetura RISC e de uma arquitetura derivada dela, a MIPS (*Microprocessor without Interlocked Pipeline Stages*).

Relembrando o contexto no qual você é um experiente profissional que trabalha no desenvolvimento de sistemas embarcados, você foi incumbido de uma nova tarefa muito importante: treinar um novo funcionário sobre as arquiteturas existentes de processadores. Esse novo colega de trabalho irá trabalhar com você no desenvolvimento, assim, é importante que ele esteja apto a realizar as mesmas funções que cabem a seu cargo. Como você possui um grande conhecimento sobre as arquiteturas de microprocessadores, que são importantes na seleção de um modelo a ser utilizado em um projeto, você irá iniciar o seu treinamento por esse aspecto.

Em um primeiro momento de estudo você optou por apresentar as principais características da arquitetura RISC, que é a mais comum nos processadores presentes em microcontroladores. Quais são as principais características dessa arquitetura? Como as instruções são executadas?

Para responder a essas questões fundamentais sobre a arquitetura RISC, você será levado a estudá-la a fundo para que seja capaz de entender por que ela é largamente utilizada em microcontrolador.

Vamos iniciar os estudos?

Não pode faltar

Todas as instruções comprehensíveis para um microcontrolador ou microprocessador digital são chamadas de conjuntos (*set*) de instruções. Quando um programa é gravado em um controlador, em linguagem *assembly*, ele especifica todas as instruções na ordem em que devem ser executadas. Essas instruções podem ser, por exemplo, para carregar um dado da memória em um registrador de uso geral ou realizar uma operação de soma dos valores presentes em dois registradores. A definição da quantidade e função dessas instruções dependem da arquitetura que o sistema digital possui. Basicamente, os conjuntos de instruções são divididos em duas grandes categorias, RISC (*Reduced Instruction Set Computer*) e CISC (*Complex Instruction Set Computer*).

A principal característica das máquinas RISC é que são constituídas por um pequeno conjunto de instruções simples executadas diretamente pelo hardware. Porém, existem outros elementos que formam a base da arquitetura RISC, são eles:

- Conjunto pequeno de instruções com largura fixa.
- Otimização na execução de chamadas de funções.
- Quantidade pequena de modos de endereçamento.
- Uso de *pipelining* em larga escala.
- Execução de uma instrução por ciclo de máquina.

Tais elementos são discutidos individualmente a seguir, trazendo os principais termos e características de cada um deles.

O ponto de partida para o desenvolvimento da arquitetura RISC foi criar um sistema com uma quantidade menor de instruções, com execução otimizada, a fim de gerar um melhor desempenho computacional. Tal consideração foi feita mesmo sabendo que uma menor quantidade de instruções possa vir a gerar programas mais longos. Portanto, a característica mais marcante da arquitetura RISC é possuir um **conjunto de instruções pequeno**, menor que o das máquinas CISC.

O fato de se ter um conjunto reduzido de instruções, além de gerar um melhor desempenho, traz outras vantagens, como: menor tamanho físico do chip e menor custo devido à necessidade de uma menor quantidade de transistores para a CPU; utilização de um decodificador de instruções com nível de complexidade reduzido, diminuindo, assim, o tempo de decodificação; redução na quantidade de bits para indicar o código de operação da instrução, o que reduz o tamanho dos programas (MONTEIRO, 2007).

Aliado ao conjunto reduzido de instruções, na arquitetura RISC, essas instruções possuem tamanho fixo, facilitando o trabalho de busca da instrução. Isso ocorre, pois a busca passa a ser realizada em uma única operação e não há necessidade de verificação do seu tamanho para o incremento do contador de instruções, já que ele será sempre igual.

Exemplificando

A título de exemplo, podemos citar como arquitetura baseada em CISC, a VAX, que possuía até 300 instruções, que podia se apresentar entre 4 bytes, 8 bytes, ou até 57 bytes. Já a arquitetura SPARC baseada em RISC possui menos de 65 instruções e todas com 4 bytes de largura (MONTEIRO, 2007).

Na execução de um programa, usualmente aparecem chamadas de rotinas e passagens de parâmetros. Essas chamadas de funções normalmente requerem poucos dados, porém realizam vários acessos à memória em operações de escrita e leitura que somadas se tornam demoradas. Para otimizar esse processo, na máquina RISC, essas chamadas são realizadas basicamente dentro do processador, que possui uma quantidade maior de registradores, evitando o acesso constante à memória externa. Dessa forma, essa **otimização na execução de chamadas de funções** melhora o desempenho total do processador, já que elas ocorrem em quantidade considerável na maioria dos programas.

A fim de simplificar o projeto e a implementação das instruções, impactando na redução da quantidade de ciclos de máquina necessários para sua execução, a arquitetura RISC utiliza **poucos modos de endereçamento** para acesso da memória. Assim, geralmente uma máquina RISC possui somente dois tipos de instruções para acesso à memória: uma tem como função transferir o dado de um dos registradores do processador para a memória (STORE) e a outra realiza o processo inverso, transferindo o dado da memória para um registrador (LOAD) (MONTEIRO, 2007).

A técnica de *pipelining* tem como objetivo principal realizar uma segmentação de instruções, fazendo com que o processador faça a execução de uma instrução, subdividindo-a em etapas. Assim, é possível que o processador execute várias instruções quase que totalmente em paralelo, aumentando o seu desempenho e reduzindo o tempo de execução das

instruções. Tendo isso em vista, a arquitetura RISC faz o uso de *pipelining em larga escala*, que é obtido devido ao formato simples e único que as instruções possuem.

Uma das características mais importantes da arquitetura RISC é a **execução de uma instrução por ciclo de máquina**, que se torna um dos requisitos essenciais para ela. Um ciclo de máquina é o tempo necessário para que sejam obtidos dois operandos dos registradores, para que seja executada uma operação na ULA (Unidade Lógica Aritmética) e para que se apresente o resultado armazenado em um registrador. Dessa forma, as instruções simples de ciclo único podem ser embutidas no hardware, tornando a sua execução mais rápida do que em outras máquinas que necessitam acessar um microprograma de controle durante a execução da instrução (STALLINGS, 2017).

Dentre os chips que utilizam arquitetura RICS, um dos primeiros a surgir foi o **MIPS**, que possui três modelos: o R2000, o R3000 e o R4000. Eles têm basicamente a mesma construção e conjunto de instruções, porém o R4000 utiliza 64 bits para os barramentos de dados internos e externos, endereçamento, registradores e ULA, ao passo que os demais utilizam 32 bits. A seguir, serão apresentadas mais características do MIPS R4000 a fim de consolidar o estudo sobre a arquitetura RISC.

O processador MIPS R4000 é dividido em duas grandes partes, uma com a CPU e outra com um coprocessador para o gerenciamento da memória. A intenção de projeto foi criar um sistema no qual a lógica de execução fosse altamente simplificada, disponibilizando espaço para lógicas de melhoria de desempenho.

Todas as instruções desse processador são codificadas em uma palavra de 32 bits, na qual as operações com memória são somente LOAD/STORE e as operações de dados são de registrador para registrador. O tamanho fixo das instruções simplifica a sua leitura e decodificação. Além disso, não existem códigos condicionais, evitando a existência de uma lógica especial para trabalhar com eles. Existem três formatos de instrução, conforme apresentado na Figura 3.12, que compartilham a mesma formação para os *opcodes* e referências a registradores. Analisando a figura, é possível identificar três tipos de instruções. A tipo I (imediato) está relacionada a operações com registradores, a tipo J (*jump*) com instruções de salto de endereço na execução e a tipo R (registrador), que pode ser utilizada para a realização de operações aritméticas (STALLINGS, 2017).

Figura 3.12 | Formatos das instruções do processador MIPS

Tipo I (immediato)	6	5	5	16				
	Operação	rs	rt	Immediato				
Tipo J (jump)	6	26					Alvo	
	Operação	Alvo						
Tipo R (registrador)	6	5	5	5	5	6	Deslocamento	Função
Operação	Código de operação							
rs	Especificador de registrador de origem							
rt	Especificador de registrador de origem/destino							
Immediato	Deslocamento imediato, desvio ou endereço							
Alvo	Endereço alvo do salto							
rd	Especificador de registrador de destino							
Deslocamento	Quantidade de deslocamento							
Função	Especificador de função de deslocamento/ALU							

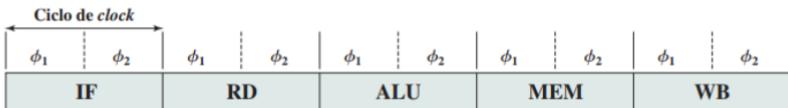
Fonte: Stallings (2017, p. 478).

Assimile

Opcode é o termo em inglês equivalente a código de operação, que se refere ao comando que o processador possui para conseguir realizar determinada tarefa. Portanto, tarefas diferentes possuem *opcodes* diferentes, que são equivalentes a códigos binários diferentes. Tais códigos são decodificados pelo decodificador de instrução, que indica à ULA qual operação realizar.

A utilização de um *pipeline* de instruções na arquitetura RISC é uma característica fundamental e isso não é diferente na máquina MIPS. Analisando inicialmente o processador MIPS R3000, encontram-se cinco estágios de *pipeline*: a busca por instrução (IF); a busca por operandos (RD); a operação da ULA ou geração de endereço dos operandos; a referência a dados de memória (MEM) e a atualização do banco de registradores (WB). Os estágios citados estão apresentados graficamente na Figura 3.13, por meio da qual é possível perceber que um ciclo de *clock*, que para essa máquina é de 60ns, é dividido em dois estágios, ϕ_1 e ϕ_2 .

Figura 3.13 | *Pipeline* simplificado do processador MIPS R3000



Fonte: Stallings (2017, p. 479).

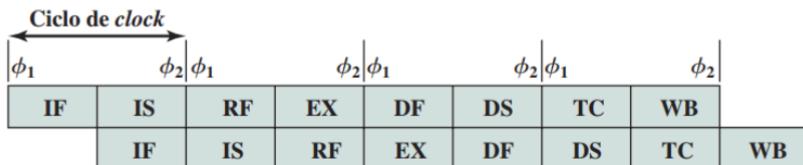
As instruções a serem executadas pelo processador são divididas conforme os estágios de *pipeline* em diferentes operações. Algumas delas requerem 60ns para serem processadas e outras, mais simples, somente 30ns. Somado a isso, existem algumas operações que são sobrepostas, tornando o *pipeline* altamente eficiente.

Assimile

Em uma máquina RISC ou MIPS, a execução de uma instrução acontece a cada ciclo de máquina e, pelo processo de *pipeline*, um ciclo de máquina é dividido em vários estágios. Dentro desses vários estágios se alocam as diferentes operações que devem ser realizadas pela CPU para execução da instrução. Tais operações podem ser executadas em um ou meio ciclo de *clock*, de forma individual ou em paralelo com outras operações. Assim, torna-se claro que, para a execução de uma instrução, são necessários vários ciclos de *clock*.

Um dos avanços que o MIPS R4000 possui em relação ao R3000 é a utilização de uma tecnologia que permite a utilização de um ciclo de *clock* de 30ns, metade do tempo de seu antecessor, bem como um acesso ao banco de registradores duas vezes mais rápido. Além disso, para o R4000 foi projetado um somador maior e mais específico, tornando possível executar operações da ULA com o dobro da velocidade. Outras otimizações também permitiram que as leituras e gravações fossem feitas com o dobro da velocidade. Somado a isso, houve um aumento da densidade do chip, possibilitando que o cache de instruções e de dados fossem incorporados a ele. Todas as melhorias citadas permitiram que o R4000 possuísse velocidade e densidade suficiente para permitir um *superpipeline* de oito estágios, com dois estágios a cada ciclo de *clock*, como está apresentado na Figura 3.14.

Figura 3.14 | *Pipeline* do processador MIPS R4000



Fonte: Stallings (2017, p. 481).

Como é perceptível na Figura 3.14, todas as operações são executadas em paralelo uma com uma outra, o que aumenta a velocidade de processamento. Os oito estágios do *pipeline* são (STALLINGS, 2017):

- Primeira metade da busca da instrução (IF): acontece a apresentação do endereço virtual para o cache de instruções e para o TLB (*Translation Lookaside Buffer*).
- Segunda metade da busca da instrução (IS): o cache de instrução retorna a instrução e o TLB ao endereço físico.
- Busca de operandos do registrador (RF): ocorre a decodificação da instrução, a verificação de condições de bloqueio, a verificação do rótulo do cache de instruções e a obtenção dos operandos a partir do banco de registradores.
- Execução de instrução (EX): podem ocorrer três diferentes atividades: a ULA executa uma operação aritmética ou lógica, caso a instrução seja uma operação de registrador para registrador; o endereço virtual de dados é calculado no caso de uma instrução LOAD ou STORE; ou ocorre a verificação das condições de desvio e cálculo do endereço virtual para o caso de uma instrução de desvio.
- Primeiro cache de dados (DF): acontece a apresentação do endereço virtual para o cache de dados e TLB.
- Segundo cache de dados (DS): o cache de dados retorna a instrução, e o TLB gera o endereço físico.
- Verificação de rótulos (TC): ocorre a verificação dos rótulos do cache para leitura e escrita.
- Atualização do banco de registradores (WB): é realizada a atualização do banco de registradores com o resultado da instrução.

Conhecendo o *pipelining* do MIPS, é possível dizer que ele apresenta todas as características de uma máquina RISC, assim, por ser um dos primeiros

chips a serem projetados com base nessa máquina, fica claro que, com o passar dos anos, surgiram diversos outros projetos. Essa evolução culminou nos microprocessadores e microcontroladores que temos atualmente.

Reflita

Nos dias de hoje, a arquitetura RISC está presente em quais tipos de dispositivos digitais? Ela é mais utilizada que a CISC?

Nesta seção você estudou as características da arquitetura RISC e pôde compreender sua forma de operação. Além disso, foi apresentado ao exemplo de um chip baseado em RISC, o MIPS, por meio do qual você foi levado a compreender como os fundamentos da arquitetura podem ser aplicados para a construção de um processador.

Sem medo de errar

Prezado aluno,

Você é um experiente profissional que trabalha no desenvolvimento de sistemas embarcados e que deve treinar um novo funcionário, apresentando as principais características da arquitetura RISC.

Embora essa arquitetura, como o próprio nome diz, paute-se em um conjunto de instruções reduzido, ela não se limita a somente isso. Dessa forma, você também deve dizer para o novo colaborador as seguintes características: conjunto pequeno de instruções com largura fixa; otimização na execução de chamadas de funções; quantidade pequena de modos de endereçamento; uso de *pipelining* em larga escala; execução de uma instrução por ciclo de máquina.

Todas as características citadas foram desenvolvidas a fim de otimizar o processamento da máquina, reduzindo o tempo de execução das instruções e aumentando o seu desempenho. Contudo, uma delas, relacionada à forma que as instruções são executadas, merece atenção especial, o *pipelining*.

Por meio do *pipelining* a execução de uma instrução é segmentada em diversos estágios, criando diversas operações que podem ser executadas em até meio ciclo de *clock*. Além disso, o *pipelining* permite o paralelismo na execução das operações. Essas técnicas permitem um acréscimo considerável na velocidade de operação do processador.

Um exemplo de máquina RISC que pode ser apresentada no treinamento é a MIPS, que foi uma das primeiras a serem desenvolvidas utilizando como base

essa arquitetura. O processador MIPS possui todas as características RISC, com funcionalidades e melhorias que mostram como essa arquitetura é importante.

Após apresentar diversas informações sobre a arquitetura RISC para o novo colaborador, ele se mostrou interessado em saber quais dispositivos atuais são implementados baseados nessa arquitetura. Você sabe que os microcontroladores atuais pautam-se na arquitetura RISC, porém, para responder a essa questão com propriedade, a sugestão é que você faça uma pesquisa a fim de dar outros exemplos de dispositivos.

Avançando na prática

Operações registrador-memória em uma máquina RISC

Você é um consultor de desenvolvimento de sistemas embarcados, na área de sistemas computacionais, e foi convidado a realizar uma palestra para um grupo de estudantes da área de computação e engenharia em um evento de uma universidade de sua cidade. O tema da palestra é sobre as principais arquiteturas de processadores baseadas nos conjuntos de instruções RISC e CISC. Assim, no dia do evento, você falou sobre essas arquiteturas, dando exemplos de dispositivos que as utilizam e, ao final da palestra, você abriu um momento de perguntas. Um aluno ouvinte da palestra lhe fez a seguinte pergunta:

— Existe alguma abordagem na máquina RISC para minimizar as operações registrador-memória?

Resolução da situação-problema

Para responder ao questionamento feito pelo aluno, você deve lembrá-lo de que uma máquina RISC possui uma quantidade mínima de instruções para o acesso à memória, sendo basicamente instruções de LOAD e STORE, onde acontece transferência de um dado da memória para um registrador e vice-versa. Essa medida torna o acesso à memória rápido, porém pode ser um problema quando muitos acessos são necessários.

Para minimizar essas operações de acesso registrador-memória, são possíveis duas abordagens básicas, uma baseada em software e outra em hardware. A abordagem de software apoia-se no compilador, que converte o programa feito em linguagem de alto nível para linguagem de máquina, a fim de que se maximize o uso de registradores. Assim, o compilador tentará

alocar registros para as variáveis que serão mais usadas em um determinado período de tempo. Contudo, essa abordagem requer o uso de algoritmos sofisticados de análise de programas. A abordagem de hardware é simplesmente para que se use mais registradores, para isso, mais variáveis podem ser mantidas em registradores por períodos mais longos.

Apresentadas as devidas explicações, o aluno o agradeceu e você finalizou a palestra.

Faça valer a pena

1. Considere o trecho a seguir sobre uma característica presente na arquitetura RISC:

“Essa técnica tem como objetivo principal realizar uma segmentação de instruções, fazendo com que o processador faça a execução de uma instrução, subdividindo-a em etapas. Assim, é possível que o processador execute várias instruções quase que totalmente em paralelo, aumentando o seu desempenho e reduzindo o tempo de execução das instruções”.

Assinale a alternativa que indica o nome da técnica citada no trecho, que é utilizada em larga escala em uma máquina RISC.

- a. Multiplexação.
- b. Indexação multusuário.
- c. Translação de dados.
- d. *Pipelining*.
- e. Operação em ponto flutuante.

2. A principal característica de um processador desenvolvido com base na arquitetura RISC é possuir um conjunto reduzido de instruções. Tal característica traz diversas vantagens para o processador, que refletem na sua velocidade de processamento, gerando um aumento do desempenho.

Assinale a alternativa que indica uma vantagem de se ter um conjunto reduzido de instruções.

- a. Aumento do tamanho físico do chip devido à necessidade de mais transistores para a CPU.

- b. Utilização de um decodificador de instruções com nível de complexidade reduzido.
- c. Aumento na quantidade de bits para indicar o código de operação da instrução, aumentando o tamanho do programa.
- d. Utilização de um sinal de *clock* baixo para limitar a velocidade do processador.
- e. Redução geral na estrutura lógica de processamento, acarretando em um alto consumo energético.

3. A principal característica das máquinas RISC é o fato de serem constituídas por um pequeno conjunto de instruções simples, que são executadas diretamente pelo hardware. Além dessa característica, é possível citar outras, como:

- () Otimização na execução de chamadas de funções.
- () Grande quantidade de modos de endereçamento.
- () Uso de *pipelining* em larga escala.
- () Execução de uma instrução por ciclo de máquina.

Classifique as alternativas em verdadeiro (V) ou falso (F) e assinale a alternativa que indica a classificação correta, respectivamente, de cima para baixo.

- a. V – V – V – V.
- b. F – V – V – V.
- c. V – F – V – V.
- d. V – F – F – V.
- e. F – F – V – V.

Seção 3

Arquitetura CISC

Diálogo aberto

Prezado aluno,

A arquitetura de um microcontrolador, de um microprocessador ou até de um simples processador pode ser baseada em um conjunto de instruções de operação simples (RISC - *Reduced Instruction Set Computer*) ou complexa (CISC - *Complex Instruction Set Computer*). Como você já conhece a arquitetura RISC, que foi trabalhada na seção anterior, esta seção dedica-se aos estudos da arquitetura CISC.

Atualmente, devido ao fato da arquitetura CISC possuir um vasto conjunto de instruções, ela está presente nos processadores para computadores que possuem um poder de processamento maior que os microprocessadores encontrados em sistemas embarcados, os quais se baseiam majoritariamente na arquitetura RISC. Portanto, é fundamental que você conheça as características de cada uma dessas arquiteturas.

No contexto que estamos trabalhando, você é um experiente profissional, que trabalha no desenvolvimento de sistemas embarcados. Sua tarefa é continuar o treinamento de um novo colaborador que entrou para sua equipe. Você já apresentou a ele as características da arquitetura RISC, agora irá trabalhar a teoria relacionada à arquitetura CISC.

As duas principais arquiteturas de processadores que podem ser encontradas são a RISC e a CISC, sendo comum a comparação entre elas. É justamente sob esse aspecto que você irá apresentar a arquitetura CISC para o seu novo colega de trabalho. Quais as características dessa arquitetura? Quais as diferenças entre a RISC e a CISC? Qual delas é a mais utilizada nos processadores atuais?

Para que você consiga responder a essas perguntas, estudará a fundo a arquitetura CISC, conhecendo os conceitos de microprogramação e microinstruções, que são largamente utilizados nessa arquitetura. Além disso, você irá comprovar a aplicabilidade dessa arquitetura estudando exemplos de máquinas que são baseadas nela. Você está preparado? Vamos lá?

Tenha um excelente estudo!

Não pode faltar

Para iniciar os estudos, é importante saber que os processadores construídos com arquitetura CISC (*Complex Instruction Set Computer*) são carregados de centenas de instruções diferentes, podendo apresentar uma elevada complexidade. Com isso, os projetos se tornam extremamente versáteis. Contudo, para a execução de algumas instruções, são utilizados microprogramas, gravados internamente no processador, que permitem receber essas instruções e executá-las por meio da utilização de várias outras instruções contidas nessa microprogramação. Seria como quebrar as instruções em diversas outras mais próximas da execução do hardware (MACHADO; MAIA, 2013).

Essa definição inicial abre caminho para o estudo dos pormenores associados à arquitetura CISC. O primeiro ponto fundamental dessa arquitetura é a utilização de um complexo e vasto conjunto de instruções com múltiplos modos de endereçamento. Tem-se a ideia de que essa diretiva advém da época inicial da computação, na qual a memória possuía um alto valor e era pequena. Assim, os códigos gerados pelos compiladores deveriam ser compactos e eficientes em sua execução. Dessa forma, cada instrução do processador deveria realizar várias funções para que o programa completo possuísse poucas instruções, reduzindo, assim, o espaço ocupado na memória (MONTEIRO, 2007).

O projeto de instruções complexas foi facilitado com o surgimento da microprogramação, o que permitiu que elas fossem implementadas em um microcódigo e que trouxessem algumas vantagens. Como o microcódigo fica armazenado na memória de controle, ele pode ter sua execução acelerada com a utilização de memórias de rápido acesso. Devido ao fato de as células de memórias serem sempre múltiplas de dois, usualmente sobra espaço nas memórias para a adição de novos microprogramas, portanto a criação de novas instruções não gera impacto sobre o espaço do chip.

Exemplificando

Um exemplo clássico de adição de novas instruções está na arquitetura x86, criada pela Intel e baseada em CISC. Nessa arquitetura, conforme novos modelos de processadores eram lançados, eram acrescentadas novas instruções sem a alteração do projeto básico original.

A arquitetura x86 está presente até hoje, com as devidas atualizações, nos processadores para computadores e servidores fabricados pela Intel, desde os modelos mais simples como a linha Celeron até os modelos avançados da linha Core i. Uma das atualizações mais relevantes que a arquitetura x86 teve foi a inclusão de um núcleo RISC dentro do processador, que aconteceu a partir do modelo 486. Assim, o núcleo RISC

executa as instruções mais simples ao passo que o núcleo CISC interpreta as instruções mais complexas, criando uma abordagem híbrida. Isso permite um desempenho global otimizado e permite que softwares antigos sejam executados sem modificação (TANENBAUM; AUSTIN, 2013).

Com isso, é possível elencar três aspectos que são considerados no desenvolvimento de um processador baseado na arquitetura CISC: a utilização de microcódigo, a construção de instruções completas e eficientes e o desenvolvimento de instruções com complexidade semelhante à dos comandos de alto nível. Tais aspectos permitem a criação de instruções, que realizam a comparação de valores e trocas de operando ou que retornam e restauram um código, por exemplo.

Mesmo que existam diversas arquiteturas baseadas em CISC, ainda que com algumas diferenças entre si, o desenvolvimento de uma máquina nessa arquitetura usualmente segue algumas regras. Tais regras, que evidenciam as características da arquitetura, são as seguintes:

- Instruções com campos de origem e destino.
- Uso de três tipos de modos: registrador para registrador; registrador para memória e memória para registrador.
- Presença de diferentes modos de endereçamento para memória.
- Instruções com largura variável e quantidade de bytes variando de acordo com o modo de endereçamento.
- Necessidade de múltiplos ciclos de *clock* para a execução de uma instrução, que varia conforme a largura da instrução.
- Hardware com poucos registradores devido à existência de muitas instruções com acesso à memória e à limitação do espaço no chip para a memória de controle (microcódigo).
- Presença de registradores especializados, como um registrador para o controle, de segmento, para o ponteiro da pilha, entre outros.

Cronologicamente, a arquitetura CISC foi criada antes da RISC. Assim, os criadores da arquitetura RISC pontuaram alguns problemas da arquitetura CISC, o que motivou a criação de um grande número de instruções, as quais acabaram por prejudicar o desempenho geral do processador. Tais pontos problemáticos estão citados a seguir (MONTEIRO, 2007):

- A existência de diferenças de velocidade entre a memória e processador, o que era um problema devido à velocidade de acesso da memória ser lenta comparada à velocidade de operação do processador.

- Devido à facilidade e à flexibilidade, os projetistas preferiram empregar o microcódigo na criação de mais instruções a realizar a programação diretamente no hardware.
- O desenvolvimento acelerado de linguagens de alto nível conduziu os projetistas de processadores a incluírem cada vez mais instruções a fim de manter um suporte adequado na compilação.
- Um vasto conjunto de instruções, com instruções próximas do significado do comando de alto nível, gera um código executado denso e compacto com poucas instruções de máquina. Porém, isso ocasiona uma maior quantidade de bits nas instruções, o que, no final das contas, não gera um código tão compacto quanto parece.
- A necessidade de compatibilidade com processadores anteriores faz com que as instruções presentes em um modelo anterior do processador sejam mantidas, acrescentando-se somente as novas, aumentando o volume total de instruções.

Um exemplo de utilização da arquitetura CISC que pode ser citado é a máquina Mic-1, que possui caráter didático e é apresentada em Tanenbaum e Austin (2013). O estudo dessa máquina é importante pois traz um exemplo de utilização da microarquitetura e da microprogramação. Ela possui uma ULA (Unidade Lógica Aritmética) ligada a dois barramentos, um sempre relacionado ao registrador H (barramento A) e outro que pode receber dados de qualquer outro registrador (barramento B), como pode ser visto na Figura 3.15.

Dentro da arquitetura, os quatro primeiros registradores (MAR, MDR, PC e MBR) operam diretamente com endereços da memória e os demais recebem dados de outros registradores pelo barramento C e enviam dados ao barramento B. É importante notar que em cada ciclo de operação os componentes da arquitetura *Mic* são controlados por sinais de controle e um microcódigo é definido como sendo o conjunto desses sinais. Ao se criar um conjunto de microcódigos, tem-se um microprograma, que é armazenado em uma memória do processador.

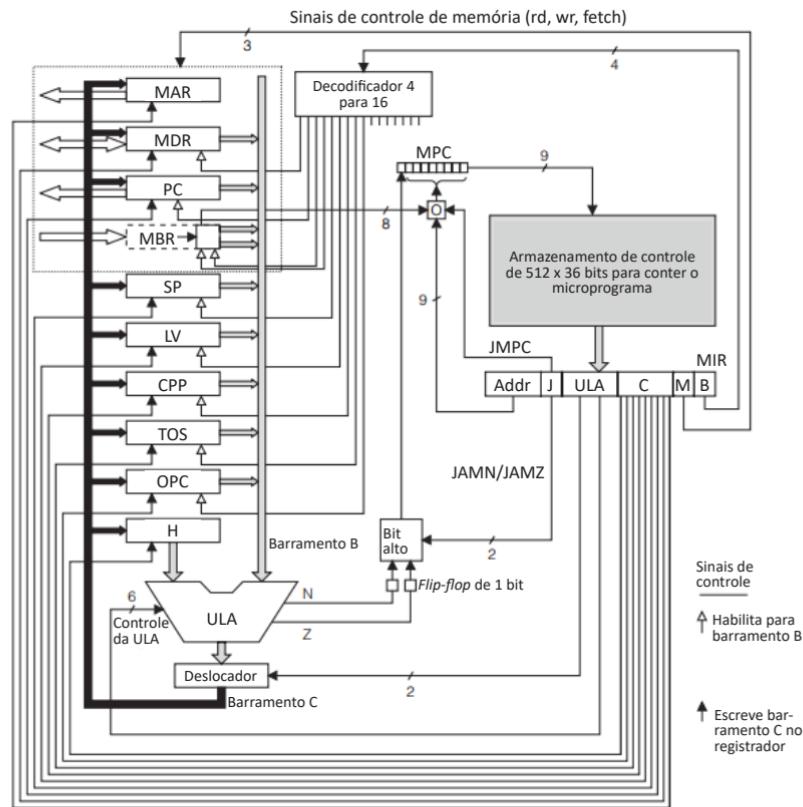
Assimile

Para controlar os dados a serem processados, é necessário um conjunto de sinais de controle que atua sobre os componentes do processador. Os valores que esses sinais de controle possuem especificam as operações a serem executadas em um ciclo do caminho de dados. O agrupamento desses sinais forma uma microinstrução, a qual é, portanto, uma instrução de máquina que faz parte de um microcódigo. Assim, um microcódigo pode possuir uma ou mais microinstruções, indicando

uma lista de pequenos passos de programa. Dessa forma, um microprograma é um conjunto de microcódigos composto por microinstruções.

Dentro da arquitetura do Mic-1, o microprograma é armazenado em uma memória chamada armazenamento de controle, que é acessada pelos registradores MBR e MDR. Essa memória contém 512 palavras, cada uma com uma microinstrução de 36 bits. Além disso, ela é acessada de uma forma diferente da memória principal, que armazena o programa que está sendo executado pelo processador em ordem de endereço. No caso, os microprogramas presentes no armazenamento de controle são mais flexíveis e são executados na sequência determinada pelas microinstruções, conforme o microcódigo que está sendo executado (TANENBAUM; AUSTIN, 2013).

Figura 3.15 | Diagrama de blocos da arquitetura Mic-1



Fonte: Tanenbaum e Austin (2013, p. 198).

Durante as últimas décadas, as universidades e centros de pesquisa estão debatendo a utilização da arquitetura RISC e CISC na construção de processadores. Assim, atualmente ainda se encontram processadores construídos pela utilização de uma ou outra arquitetura, ou ainda uma estrutura híbrida. Dessa forma, é importante saber diferenciar claramente as características dessas duas arquiteturas.

Ambas apresentam vantagens e desvantagens e a escolha do modelo adotado vai depender do tipo de aplicação. O Quadro 3.2 apresenta um comparativo entre as duas abordagens.

Quadro 3.2 | Comparação entre arquiteturas RISC e CISC

Características	RISC	CISC
Complexidade e quantidade de instruções	Instruções simples e em número reduzido.	Muitas instruções complexas.
Tamanho das instruções	Instruções de tamanho fixo.	Instruções de tamanho variável.
Complexidade	Complexidade no compilador.	Complexidade no código.
Modos de endereçamento	Poucos modos de endereçamento.	Muitos modos de endereçamento.
Quantidade de ciclos	Instruções em um único ciclo de máquina.	Instruções em múltiplos ciclos de máquina.
Paralelismo	Altamente paralelizado.	Fracamente paralelizado.
Acesso aos dados	Acesso aos dados via registradores.	Acesso aos dados via memória.
Microprogramação	Execução sem microprogramação.	Execução com microprogramação.
Quantidade de registradores	Vários registradores.	Poucos registradores.

Fonte: adaptado de Machado e Maia (2017, p. 31).

Basicamente, os controladores RISC utilizam unidades de controle mais simples, rápidas e baratas, optando pelas instruções mais simples possíveis, com pouca variedade e com poucos endereços. Como consequência, os circuitos internos também são mais simples e conseguem trabalhar com *clocks* mais altos. Outra vantagem das máquinas RISC é a previsibilidade. Pelo fato de as instruções variarem pouco de uma para a outra, a unidade de controle consegue prever quantos ciclos de máquina serão necessários para executar determinada função, acarretando um ganho de desempenho no

processamento paralelizado. Assim, máquinas RISC geralmente resultam em projetos menores, mais baratos e que consomem menos energia. Isso os torna muito interessantes para dispositivos móveis e aplicações em tempo real.

Já as máquinas com conjuntos de instruções CISC investem em unidades de controle poderosas e capazes de executarem tarefas complexas. Por apresentarem instruções gravadas no processador, há uma facilidade no trabalho dos programadores, além de reduzirem o código executável. São máquinas maiores, consomem mais energia, são mais caras, porém o processamento é mais poderoso.

O conjunto CISC geralmente está associado à arquitetura Von Neumann, enquanto a arquitetura Harvard se liga ao modelo RISC. Contudo, esses modelos estão cada vez mais similares, pois as máquinas RISC já apresentam um conjunto maior de instruções, equivalentes aos primeiros modelos CISC. O aumento da tecnologia dos semicondutores também já proporcionou a processadores CISC a possibilidade de executarem mais de uma instrução por ciclo de máquina, alcançando velocidades semelhantes a dos equipamentos desenvolvidos em RISC.

Quanto à aplicabilidade, atualmente os dispositivos que possuem processadores ou microprocessadores com arquitetura RISC são basicamente sistemas embarcados. Várias arquiteturas de microcontroladores, como a AVR, presente na plataforma de desenvolvimento Arduino, são baseadas em uma máquina RISC, a qual serviu de base para o desenvolvimento da arquitetura ARM (originalmente *Acorn RISC Machine* e depois *Advanced RISC Machine*), que constitui desde microcontroladores até processadores para smartphones. A título de curiosidade, os processadores para smartphone Snapdragon, fabricados pela Qualcomm, os processadores Helio, fabricados pela MediaTek, e os processadores Exynos, fabricados pela Samsung, são todos baseados na arquitetura ARM.

Atualmente a arquitetura CISC fica limitada a processadores utilizados em computadores, com arquitetura derivada da x86. Ela está presente nos processadores fabricados pela Intel, como já foi citado, e também nos fabricados pela AMD, desde a linha Athlon (mais simples) até a linha Ryzen, com maior capacidade de processamento.

Reflita

Conhecendo agora as características das arquiteturas RISC e CISC, na sua opinião, qual delas proporciona o melhor desempenho? Isso é válido para todos os tipos de processadores, sejam eles de grande ou de pequena capacidade de processamento?

Sem medo de errar

Você é um experiente profissional, que trabalha no desenvolvimento de sistemas embarcados e está realizando um treinamento de um novo colaborador que entrou para sua equipe. Você deve apresentar a ele as características da arquitetura CISC, comparando-a à arquitetura RISC.

Dentre as principais características da arquitetura CISC, pode-se mencionar a utilização de microcódigo, a construção de instruções completas e eficientes e o desenvolvimento de instruções com complexidade semelhante à dos comandos de alto nível. Aliado a essas características, podem ser elencadas algumas regras para o desenvolvimento de uma máquina CISC, como a presença de diferentes modos de endereçamento para a memória, instruções com largura variável e quantidade de bytes variando de acordo com o modo de endereçamento, hardware com poucos registradores e a necessidade de múltiplos ciclos de *clock* para a execução de uma instrução, por exemplo.

A arquitetura CISC possui diversas diferenças em relação à RISC. Isso ocorre, pois, como a arquitetura RISC foi desenvolvida depois, ela tomou como base as deficiências da arquitetura CISC a fim de criar uma máquina mais eficiente. Assim, a arquitetura CISC apresenta muitas instruções complexas e de tamanho variável, muitos modos de endereçamento, instruções sendo executadas em múltiplos ciclos de máquina e poucos registradores. Já a arquitetura RISC utiliza poucas instruções simples de tamanho fixo, poucos modos de endereçamento, vários registradores internos ao processador e as instruções são executadas em um único ciclo de *clock*.

Atualmente a arquitetura CISC serve como base para os processadores de alto poder de processamento, utilizados em sistemas computacionais como computadores e servidores. A arquitetura RISC está presente nos sistemas embarcados, como em microprocessadores e microcontroladores. Porém, vale ressaltar que já se tentou utilizar processadores baseados em CISC em um sistema embarcado (smartphone) e processadores RISC em computadores. Contudo, devido à falta de compatibilidade dos sistemas operacionais, esses usos não perduraram. Talvez, futuramente, essa barreira seja superada e se encontrem processadores derivados de ambas arquiteturas em todas as plataformas.

Comparativo de funcionamento de uma máquina RISC e uma CISC

Você é um consultor na área de sistemas computacionais e está participando de um *workshop* com diversos profissionais da área. Eles possuem muitos conhecimentos práticos no desenvolvimento de sistemas embarcados e softwares para computador, porém existem alguns conceitos que eles não conseguem compreender. Todos eles conhecem as características das máquinas RISC e CISC, mas não conseguem entender as diferentes características na prática. Assim, foi pedido a você que apresentasse um exemplo das diferentes abordagens que as máquinas possuem. Qual exemplo apresentar?

Resolução da situação-problema

De início é necessário deixar claro que a principal diferença entre as arquiteturas RISC e CISC é o número de instruções presentes no processador: a máquina CISC possui um conjunto amplo e complexo e a máquina RISC um conjunto simples e reduzido. O seu exemplo vai se basear justamente nessa diferença fundamental.

Vamos considerar a execução de uma instrução de soma com dois operandos. Um deles está armazenado em um registrador (reg1) e outro armazenado na memória, no endereço X. Para realizar a soma, independentemente da arquitetura, ela é executada entre dois registradores internos ao processador (reg1 e reg2).

Assim, uma máquina CISC irá executar o seguinte comando em uma única instrução:

ADD reg1,reg2 X

Essa instrução realiza o acesso à memória no endereço X, retira o valor ali contido e o transfere para o registrador reg2; por fim, realiza a soma entre reg1 e reg2.

Em uma máquina RISC, a execução acontece da seguinte forma:

MOV reg2,X

ADD reg1,reg2

Perceba que são executadas duas instruções, uma delas realiza o acesso à memória e a outra a soma entre os registradores. Nesse caso, a máquina RISC executou o dobro das instruções que a CISC e essa diferença pode ser ainda maior para as outras instruções. Porém, a simplicidade das instruções e sua pequena variabilidade pode tornar sua execução mais rápida do que uma única instrução mais complexa.

Vale ressaltar que os nomes reg1 e reg2, bem como o nome das instruções, são apenas ilustrativos e podem ser encontrados ou não em uma máquina real. Com isso, você sanou a dúvida de seus colegas profissionais.

Faça valer a pena

1. Para controlar os dados a serem processados, é necessário um conjunto de sinais de controle, que atuam sobre os componentes do processador. Os valores que esses sinais de controle possuem especificam as operações a serem executadas em um ciclo do caminho de dados.

O agrupamento desses sinais de controle forma:

- a. Uma porta lógica.
- b. Uma operação binária.
- c. Um microprograma.
- d. Um microcódigo.
- e. Uma microinstrução.

2. Os sistemas computacionais são baseados em basicamente duas arquiteturas: a RISC e a CISC, das quais derivaram muitas outras, mantendo as características que as originaram. A seguir são apresentados alguns exemplos de arquiteturas:

- I. ARM.
- II. x86.
- III. Mic-1.
- IV. MIPS.

Assinale a alternativa que indica a classificação correta das arquiteturas, conforme a arquitetura que as originou.

- a. I – RISC; II – CISC; III – CISC; IV – RISC.
- b. I – RISC; II – CISC; III – RISC; IV – RISC.
- c. I – CISC; II – RISC; III – RISC; IV – CISC.
- d. I – CISC; II – RISC; III – CISC; IV – CISC.
- e. I – RISC; II – RISC; III – CISC; IV – RISC.

3. A arquitetura CISC possui diversas características que permitem a sua diferenciação da arquitetura RISC e determinam a forma de operação de um sistema. Considere as características apresentadas a seguir:

- I. Utilização de microcódigo.
- II. Construção de instruções completas e eficientes.
- III. Conjunto de instruções simples e reduzido.
- IV. Hardware com poucos registradores.
- V. Hardware com muitos registradores.

Enunciado:

- a. Somente III e V.
- b. Somente I e II.
- c. Somente II e IV.
- d. Somente I, II e IV.
- e. Somente I, III e V.

Referências

- MACHADO, F. B.; MAIA, L. P. **Arquitetura de sistemas operacionais**. 5. ed. Rio de Janeiro: LTC, 2013.
- MACHADO, F. B.; MAIA, L. P. **Arquitetura de sistemas operacionais**. 5. ed. Rio de Janeiro: LTC, 2017.
- MICROCHIP. ATmega48A/PA/88A/PA/168A/PA/328/P. [S.l.: s.n.], 2018. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf>. Acesso em: 10 abr. 2020.
- MONK, S. **30 projetos com arduino**. 2. ed. Porto Alegre: Bookman, 2014.
- MONTEIRO, M. A. **Introdução à organização de computadores**. 5. ed. Rio de Janeiro: LTC, 2007.
- OLIVEIRA, C. V.; ZANETTI, H. P. **Arduino Descomplicado – Como Elaborar Projetos de Eletrônica**. São Paulo: Érica, 2015.
- PAIXÃO, R. R. **Arquitetura de Computadores – PC's**. São Paulo: Érica, 2014.
- PEREIRA, F. **Microcontrolador PIC18 Detalhado: hardware e software**. São Paulo: Érica, 2010. E-book.
- SOUZA, D. J. de. **Desbravando o PIC**: ampliado e atualizado para PIC16F628A. 12. ed. São Paulo: Érica, 2010. 268 p.
- STALLINGS, W. **Arquitetura e organização de computadores**. 10. ed. São Paulo: Pearson Education do Brasil, 2017.
- STALLINGS, W. Visão de Alto Nível da Função e Interconexão do Computador. In: STALLINGS, W. **Arquitetura e organização de computadores**. 10. ed. São Paulo: Câmara Brasileira do Livro: Editora Pearson, 2018.
- TANENBAUM, A. S.; AUSTIN, T. **Organização estruturada de computadores**. 6. ed. São Paulo: Pearson Prentice Hall, 2013.
- TANGON, L. G., SANTOS, R. C. dos. Componentes básicos de um computador. In: TANGON, L. G., SANTOS, R. C. dos. **Arquitetura e organização de computadores**. Londrina: Editora e Distribuidora Educacional S.A, 2017. Disponível em: <https://biblioteca-virtual.com/detalhes/livro/785>. Acesso em: 8 abr. 2020.
- VERLE, M. **PIC Microcontrollers – Programming in BASIC with examples**. [S.l.]: MikroElektronika, 2010. E-book. Disponível em: <https://www.mikroe.com/ebooks/pic-microcontrollers-programming-in-basic/introduction>. Acesso em: 13 abr. 2020.

Unidade 4

Giancarlo Michelino Gaeta Lopes

Programação de microprocessadores e microcontroladores

Convite ao estudo

Prezado aluno,

A programação de um microcontrolador é algo fundamental para o profissional que trabalha no desenvolvimento de sistemas eletrônicos, os quais em sua larga maioria são digitais. Conhecendo a organização interna dos microcontroladores, o próximo passo é compreender como é realizada a sua programação.

Nesse momento, é importante saber diferenciar um microprocessador de um microcontrolador e entender que a programação tratada nesta unidade será a de um microcontrolador, que possui memórias e diversos periféricos dentro de um único chip. Portanto, o algoritmo criado será compilado e enviado para dentro do microcontrolador, será armazenado em sua memória e executado sequencialmente. Assim, o foco dos estudos é o de como realizar a configuração e utilização de periféricos diversos para que o microcontrolador possa realizar operações de controle, bem como comparações e cálculos necessários para o processamento de dados.

Além de conhecer a programação de microcontroladores, o conhecimento de novas arquiteturas e tendências permite a implementação de novas funcionalidades ao projeto, que até pouco tempo atrás eram impossíveis de serem implementadas em um sistema embarcado. Uma dessas tecnologias é a Inteligência Artificial, que requer um poder de processamento relativamente alto. Um grande avanço em termos de processamento ocorreu com a arquitetura ARM, uma máquina RISC que permitiu um grande aumento na capacidade de processamento dos microcontroladores, trazendo dispositivos de 32 bits com um baixo custo. Dessa forma, conhecer a programação de um microcontrolador com tal arquitetura tem a sua relevância, já que é aplicada em equipamentos que exigem o processamento de um grande volume de dados.

Dentro desta unidade de estudo, a primeira seção trabalha a programação da plataforma de desenvolvimento Arduino, mostrando os diferentes tipos

existentes e as aplicações possíveis. Por sua vez, a segunda seção aborda novas arquiteturas de microcontroladores e tendências, trabalhando a organização interna da arquitetura ARM e conceitos de Inteligência Artificial. A última seção traz aplicações para microcontroladores ARM, mostrando como é realizada a sua programação e quais são suas especificidades.

Bons estudos!

Microcontrolador: Arduino

Diálogo aberto

Prezado aluno,

Com a evolução tecnológica, grande parte dos sistemas eletrônicos se tornaram digitais e passaram a ser controlados por microcontroladores. Um exemplo disso está no ambiente industrial, dominado pelos CLPs (Controlador Lógico Programável) como elementos de controle. Esses dispositivos possuem internamente um microcontrolador, que executa as instruções para as quais foi programado. Porém, o uso dos microcontroladores não se limita ao ambiente industrial. Grande parte da tecnologia com a qual temos contato no nosso cotidiano possui um microcontrolador, ou mais de um, como acontece em um automóvel que possui microcontroladores para o controle da injeção eletrônica, do sistema de freio, do alarme, de *air bags*, entre diversos outros subsistemas.

A partir do que foi dito, fica evidente que um profissional que trabalha com tecnologia deve saber programar um microcontrolador e é justamente sobre isso que esta seção trata, a programação de um microcontrolador. Dentro os modelos existentes, a plataforma Arduino merece atenção especial devido a sua simplicidade na programação e vasta gama de bibliotecas. Assim, esta seção dá destaque aos estudos da programação desse dispositivo, utilizando a linguagem de programação em C/C++ e demonstrando como é feita a configuração e utilização dos principais periféricos que um microcontrolador possui.

Relembrando o contexto em que você é um profissional desenvolvedor de sistemas embarcados, você foi incumbido de um novo projeto que está sendo realizado pela empresa onde trabalha. Uma startup procurou a sua empresa para desenvolver um dado eletrônico com LEDs, a fim de substituir o dado convencional, para ser utilizado em um brinquedo criado por eles.

Assim, você foi incumbido de realizar esse projeto e apresentar um protótipo funcional o mais rápido possível. Como o desenvolvimento deve ser ágil, você optou por utilizar a plataforma Arduino em conjunto com outros componentes eletrônicos para que o sistema se torne funcional. O requisito é que o número seja sorteado aleatoriamente a partir do apertar de um botão e que ele seja exibido graficamente de forma semelhante às faces

de um dado. Como projetar esse sistema? Qual a montagem de hardware que deve ser feita? Como elaborar o *firmware* de controle para esse sistema?

Perceba que, para desenvolver esse projeto, são necessários diversos conceitos e tudo se inicia com o conhecimento dos tipos de Arduino existentes. Sabendo isso, você será levado a estudar como realizar a programação de um Arduino utilizando todos os periféricos e funcionalidades. Vamos iniciar os estudos?

Não pode faltar

A programação de um microcontrolador passa por diversas etapas e, primeiramente, deve-se obter o conhecimento da linguagem de programação que será utilizada.

A primeira linguagem que foi utilizada para a programação de um microcontrolador foi a *assembly*, que realiza a chamada direta das instruções do processador. Em função de sua proximidade com a linguagem de máquina, sua execução é muito rápida, o que também a torna complexa. Assim, os microcontroladores passaram a utilizar a linguagem C como padrão de programação, algo que acontece até hoje.

Devido ao aumento da complexidade dos algoritmos, a linguagem C utilizada em microcontroladores passou a integrar elementos da linguagem C++, que até então era utilizada somente para o desenvolvimento de softwares computacionais. Isso acontece com a linguagem utilizada para a programação dos microcontroladores da plataforma Arduino, que utiliza uma linguagem mista entre C e C++, a qual será a base dos estudos desta seção.

A placa de desenvolvimento Arduino é formada por um microcontrolador da fabricante Microchip, circuitos de entrada/saída e que pode ser conectada facilmente em um computador para programação. Para isso, utiliza-se uma IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado), por meio da qual o código é desenvolvido, e um cabo USB transfere o programa para o controlador (OLIVEIRA, 2015). A vantagem dessa plataforma é que ela possui uma enorme quantidade de módulos chamados de *Shields*, que se encaixam na placa principal ou são conectados a uma matriz de contatos, expandindo seus recursos. Assim é possível a utilização de sensores e circuitos integrados de forma rápida.

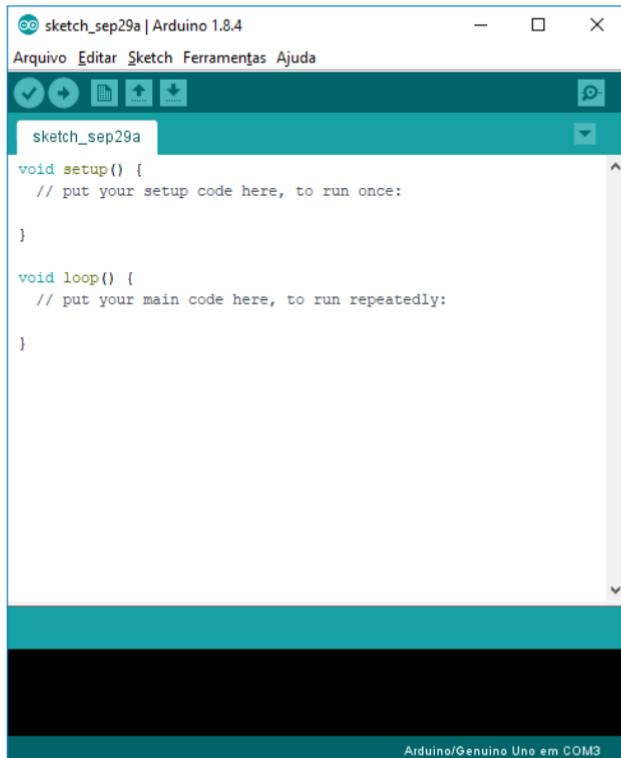
Outro ponto importante que faz com que o Arduino seja largamente utilizado atualmente é a vasta quantidade de bibliotecas disponíveis. Esse é um fator determinante no tempo de desenvolvimento de um projeto em que o projetista não se vê obrigado a desenvolver um código próprio para ler os dados de um sensor de temperatura, por exemplo, basta que ele utilize a biblioteca disponibilizada no próprio ambiente de desenvolvimento ou a obtenha na internet. Dessa forma, o desenvolvimento de soluções com a plataforma Arduino se torna ágil e confiável.

Antes de entrarmos na programação do Arduino, é importante conhecer os diferentes tipos existentes. O modelo mais difundido é o Arduino uno, que é baseado no microcontrolador ATmega328, possui catorze pinos de entrada/saída, seis entradas analógicas e opera com frequência de *clock* de 16 MHz. Com base nesse mesmo microcontrolador, mas em uma montagem mais compacta, existe o Arduino Nano. Utilizando o microcontrolador ATmega32u4, existem os Arduinos Leonardo e Micro, que possuem uma porta USB integrada, permitindo que sejam reconhecidos por mouses e teclados, por exemplo. Com uma maior quantidade de pinos e periféricos, há o Arduino Mega, baseado no microcontrolador ATmega2560.

Todos os Arduinos apresentados até aqui são constituídos de 8 bits, ou seja, possuem barramento, memória e também registradores, todos com 8 bits de largura e com direcionamento de sua aplicação para sistemas mais simples, para aplicações que demandem pequenos volumes no processamento de dados. Para um poder de processamento maior, a opção é utilizar o Arduino Due, que é baseado no microcontrolador SAM3X8E, já com núcleo ARM de 32 bits.

Para realizar a programação de um Arduino, é utilizada a sua IDE proprietária, que é disponibilizada para download gratuitamente na página oficial da plataforma de desenvolvimento. A página inicial do software (versão 1.8.5) é apresentada na Figura 4.1, na qual é possível perceber uma interface bastante intuitiva e simples para facilitar a navegação do usuário. Vale ressaltar que antes de iniciar a elaboração de códigos, são necessárias algumas configurações do software, informando qual o modelo da placa utilizada e a qual porta COM do computador o Arduino está conectado. Feito isso, o usuário pode iniciar um código do zero, ou utilizar como base um dos inúmeros exemplos que podem ser acessados facilmente.

Figura 4.1 | Ambiente de desenvolvimento Arduino- tela inicial



Fonte: captura de tela do software Arduino IDE 1.8.5.

O *sketch* do Arduino, onde é estruturado o programa, apresenta uma estrutura padronizada e é composto por duas funções obrigatórias: *Setup* e *Loop*. A função *Setup* tem o objetivo de configurar o Arduino. Portanto, nesta seção ocorre a inicialização das variáveis, a declaração dos dispositivos I/O (entradas e saídas), definem-se também as velocidades de transmissão dos periféricos de comunicação, entre outros. Assim, de uma única vez, a função *Setup* inicializa todos os códigos que estejam no início do programa. A função *Loop* consiste na parte do código que é executada infinitamente enquanto a placa estiver ligada. Ou seja, quando o processador executar a última linha de código do programa, um novo ciclo da função *Loop* é iniciado a partir da linha um. Portanto, a maioria do código elaborado pelo programador é executado dentro dessa função. Sua primeira execução ocorre após o término da função *Setup*.

Agora que já entendemos como operar o ambiente de desenvolvimento e como o *sketch* é estruturado, podemos iniciar o estudo sobre o desenvolvimento de projetos. Para isso, serão apresentados os principais periféricos que um microcontrolador possui e como eles são configurados e utilizados em um programa criado dentro da plataforma Arduino.

Um dispositivo **entrada/saída** pode ser configurado como entrada ou saída digital. No modo entrada, ele permite que o estado do pino seja monitorado e armazenado em um endereço de memória. Já no modo de saída, é possível configura-lo para que o sinal de saída apresente nível lógico alto ou baixo. Sendo assim, vamos verificar como são feitas essas configurações no Arduino.

A declaração de um pino como entrada ou saída é feita através da sintaxe: “pinMode(pino, modo)”. O parâmetro “pino” corresponde ao número do pino identificado ao lado do conector do Arduino, como pode ser visto na área destacada como digital na Figura 4.2, podendo variar de 0 a 13. Já o parâmetro “modo” determina se o pino é entrada (INPUT), entrada com resistor de pull-up (INPUT_PULLUP) ou saída (OUTPUT).

Figura 4.2 | Arduino Uno



Fonte: Wikimedia Commons .

Para a leitura do estado de um pino declarado como entrada, utiliza-se a função “digitalRead(pino)”. O valor de retorno será 1 (HIGH) caso o sinal apresente nível lógico alto e 0 (LOW) para nível lógico baixo. É preciso lembrar, ainda, que os limiares de detecção dependem do valor de alimentação do microcontrolador e são apresentados na folha de dados do componente.

Para os pinos declarados como saídas, utilizamos a função “digitalWrite (pino, valor)” para determinar o estado do pino. Quando “valor” for configurado como 1 (HIGH), a tensão de saída será de VCC (5 V ou 3,3 V). Em caso de 0 (LOW), a tensão de saída será de 0 V.

Exemplificando

O trecho de código a seguir é um exemplo de configuração do pino 13 como saída, do pino 7 como entrada e do pino 9 como entrada com resistor de *pull-up*. Além de mostrar como pode ser lido o estado do pino 7 e armazenado na variável “val”, e como pode ser colocado nível alto e baixo, no pino 13, configurado como saída:

```
1  unsigned char val;  
2  
3  void setup()  
4  {  
5      pinMode(13, OUTPUT);  
6      pinMode(7, INPUT);  
7      pinMode(9, INPUT_PULLUP);  
8  }  
9  
10 void loop ()  
11 {  
12     val = digitalRead(7);  
13     digitalWrite(13, HIGH);  
14     delay (1000);  
15     digitalWrite(13, LOW);  
16     delay (1000);  
17 }
```

Fonte: elaborado pelo autor.

No código apresentado, a função “`delay (ms)`” força o controlador a parar sua execução durante o tempo (em milissegundos) expresso em seu argumento.

Refletá

Diversos códigos utilizam a função `delay` para gerar atrasos de tempo, porém ela interrompe a execução do algoritmo. Existe outra forma de criar um atraso de tempo no Arduino?

O módulo CCP (do inglês, *Capture/Compare/PWM*) é um periférico que permite ao programador controlar e medir o tempo de diversos eventos. Assim, seu funcionamento está associado diretamente aos temporizadores e contadores, que são usados como sinais de entrada para o sistema CCP. Essa relação fica mais evidente quando se verifica que o controle de tais funções são feitos nos mesmos registradores utilizados para configurar os temporizadores, no caso dos ATmega328 (MICROCHIP, 2018). Esse microcontrolador, utilizado no Arduino Uno, possui três *timers* com módulos CCP capazes de gerarem até seis sinais de PWM nos pinos digitais indicados com um til (~), como pode ser visualizado na Figura 4.2.

Assimile

PWM é a sigla para *Pulse Width Modulation*, ou modulação por largura de pulso, que consiste em um sinal digital com frequência fixa, no qual o tempo em que o sinal fica em nível alto ou baixo pode variar. Essa variação do tempo permite realizar o controle de diversos dispositivos, modificando a potência que é aplicada sobre eles.

O modo de captura permite a medição do tempo de duração de um determinado evento, monitorando o estado de um timer (que constantemente muda de valor). Já o modo de comparação, compara o valor do timer com um valor previamente configurado, permitindo gerar ações quando o valor da contagem do temporizador igualar-se ao valor de referência. Ou seja, ele simula o estouro do temporizador em um valor determinado, sem que de fato o temporizador estoure sua capacidade de contagem. Por fim, o modo PWM gera sinais modulados por largura de pulso, permitindo ao programador escolher a frequência e o ciclo de trabalho a serem adotados, com base nos sinais gerados pelo temporizador. Todavia, por serem partes da mesma estrutura, só é possível utilizar um sistema de cada vez (PEREIRA, 2002).

O Arduino disponibiliza ainda, em seu ambiente de desenvolvimento, alguns sinais do tipo **PWM**. Conforme a função utilizada, é possível obter um sinal de saída com frequência fixa de 980 Hz para os pinos 5 e 6, enquanto que para os pinos 3, 9, 10 e 11 o valor é de 490 Hz. Essa função é executada a partir da sintaxe “analogWrite (pino, valor)”. O primeiro argumento indica o pino onde será gerado o PWM. O segundo argumento, “valor”, determina a razão cíclica do sinal PWM e é representado por uma variável de 8 bits. Portanto, são aceitos valores entre 0 e 255, em que 0 corresponde a um DC de 0% e 255 a um de 100%.

Os **conversores analógico-digitais** (A/D) são utilizados para transformar sinais analógicos, contínuos no tempo, em sinais amostrados, discretos no tempo, e quantizados dentro de um número finito de valores. Isso é feito para que o microcontrolador consiga processar informações de um sinal analógico externo que queremos.

O processo de conversão do sinal é feito em três etapas. Na primeira etapa, amostragem, o sinal é retido em um valor fixo durante um período de tempo. Assim, a cada período, coleta-se um valor de amplitude do sinal. Essa periodicidade em que as amostras são coletadas é denominada de período de amostragem. Na segunda etapa, denominada de quantização, a amplitude do sinal é dividida em regiões igualmente espaçadas. Ou seja, o sinal de amplitude também é discretizado e passa a apresentar um número finito de valores inteiros, determinados pelo número de bits do conversor A/D. Por fim, o sinal passa pela etapa de codificação em que cada valor discreto é representado por uma sequência binária.

O microcontrolador ATmega328 apresenta um único conversor A/D de 10 bits, mas que pode ser multiplexado em seis canais diferentes desde que seja utilizado um canal por vez. Assim, cada canal é associado a um dispositivo I/O, permitindo que o controlador digitalize até seis sinais de entrada diferentes. O resultado da conversão é armazenado em dois registradores visto que cada registrador apresenta 8 bits (OLIVEIRA, 2015).

Dentro da IDE Arduino, a função utilizada para realizar uma conversão é a “analogRead (pino)”, na qual o argumento “pino” corresponde a um dos seis canais de entrada (0-5). Como resposta, a função retorna um algarismo inteiro cujo valor pode variar de 0 a 1023. Caso se esteja utilizando a alimentação do Arduino em 5 V, a variação de um nível para outro é de 4,88 mV. Portanto, caso uma conversão retorne o valor de 400, significa que o valor de tensão de entrada era de aproximadamente 1,95 V.

Exemplificando

O código apresentado a seguir realiza a leitura de um valor analógico aplicado na entrada A2 de um Arduino Uno e utiliza esse valor para a geração de uma onda PWM no pino 5. Como a conversão A/D retorna um valor de 10 bits e a onda PWM permite um valor de somente 8 bits, é necessário realizar uma operação matemática para adequar o valor.

```
1 int PinoSensor = A2;
2 int PinoPWM = 5;
3 int ValorSensor;
4
5 void setup()
6 {
7     // sem inicialização
8 }
9
10 void loop()
11 {
12     ValorSensor=analogRead(PinoSensor);
13     analogWrite(PinoPWM, ValorSensor/4);
14     delay(10);
15 }
```

Fonte: elaborado pelo autor.

Perceba que as funções analógicas do Arduino não necessitam de inicialização, podendo ser utilizadas diretamente no código. Repare ainda que foi utilizada a função *delay*, cujo objetivo é aumentar a estabilidade das leituras e evitar o travamento do microcontrolador, já que as funções são executadas muito rapidamente.

Uma melhoria que poderia ser feita nesse código é a realização de alguma medida para reduzir o ruído do sinal analógico captado na conversão. A forma mais simples seria realizar a média de algumas medidas antes de aplicar o resultado para a geração da onda PWM.

Os dispositivos de **comunicação serial** são, sem sombra de dúvidas, um dos principais periféricos que um microcontrolador pode apresentar. Eles permitem a comunicação do chip com diversos outros aparelhos, dentre eles: computadores, microcontroladores, modems, memórias, conversores A/D, sensores, entre outros.

Para que toda essa variedade de componentes consiga se comunicar, um protocolo de comunicação é implementado, permitindo ao receptor interpretar a mensagem recebida. Os principais periféricos de comunicação serial de microcontroladores são: USART (*Universal Synchronous and Asynchronous Serial Receiver and Transmitter*), SPI (*Serial Peripheral Interface*) e I2C (*Inter-Integrated Circuit*).

O módulo de comunicação USART permite que o chip estabeleça comunicação com elementos externos, sem nos preocuparmos muito com a programação das rotinas de recepção e transmissão. Isso é possível graças aos circuitos internos do periférico, que são responsáveis por obedecer às características de implementação do protocolo. A principal desvantagem desse modelo de comunicação é que a conexão é feita somente entre dois dispositivos.

Esse periférico apresenta dois modos de operação: modo síncrono e modo assíncrono. No primeiro, utilizam-se três fios para a comunicação, mais o terra, sendo eles: *clock*, transmissor e receptor. Já para o segundo, o modo de operação assíncrono, utiliza-se um par de fios, mais o terra, sendo um para transmissão e outro para recepção. Isso permite um regime de comunicação *full-duplex*, em que ambos os dispositivos podem transmitir e receber dados ao mesmo tempo. Essa simplicidade de circuito faz com que o modo de operação assíncrono seja o mais utilizado.

O funcionamento desse sistema de comunicação, em sua forma assíncrona (UART), está atrelado à interpretação bit a bit pelo receptor. Cada pacote enviado contém um *start bit*, que indica o início da transmissão; um ou dois *stop bits* para indicar o fim dos dados; cinco a nove bits de informação e pode apresentar um bit de paridade, que é utilizado para verificar se a mensagem foi recebida sem erros.

Outro conceito importante é a definição do *baud rate*, que corresponde à velocidade com que os dados trafegam pela comunicação, expressos em bps (bits por segundo) (PEREIRA, 2002). Seu valor é importante pois determina a duração de tempo de cada bit, garantindo que o receptor identifique todos os bits transmitidos. Além disso, para a comunicação ser estabelecida, o pino de recepção de um dispositivo deve ser ligado ao pino transmissão do outro e vice-versa e os dois dispositivos devem possuir o mesmo terra.

O ambiente de desenvolvimento Arduino traz um conjunto de funções que permite configurar uma comunicação serial UART de maneira simples e

rápida. Dessa forma, para o Arduino Uno, o pino digital 0 passa a funcionar como receptor (Rx) e o pino digital 1 como transmissor (Tx).

Para iniciar uma comunicação, utiliza-se a sintaxe “Serial.begin (velocidade,config)”, cujos valores padrão de velocidade (*baud rate*) são: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 ou 115200. O segundo argumento é opcional e é utilizado para configurar o número de bits de dados, de paridade e de parada (*stop bits*). Caso nenhum valor seja configurado, por padrão, a comunicação apresenta oito bits de dados, nenhum bit de paridade e apenas um bit de parada. A comunicação serial é encerrada através da função “Serial.end()”.

Vamos aprender agora a enviar e receber informações através das funções de escrita e leitura. Uma das principais funções de escrita (que envia um dado para o outro dispositivo) é a função “Serial.print()”, utilizada para enviar caracteres ou *strings* no formato ASCII. Temos também a função “Serial.println()”, que possui a mesma finalidade, contudo, ao terminar o envio de dados, o cursor é avançado para a próxima linha. A recepção de dados é feita através da função “Serial.read()”, que permite a leitura de um byte de dados.

Exemplificando

Por meio das bibliotecas do Arduino, é possível que um dado seja enviado de diversas formas (Figura 4.3).

Figura 4.3 | Exemplos de utilização das funções de escrita

The screenshot shows the Arduino IDE interface. On the left, the code window contains the following sketch:

```
int a = 5;
void setup()
{
    Serial.begin(9600);

    Serial.print(78);
    Serial.print("\t");
    Serial.print('N');
    Serial.println("Hello World");
    Serial.println(78,BIN);
    Serial.println("");
    Serial.println(78,OCT);
    Serial.println(78,HEX);
    Serial.println(a);
}
```

On the right, the串行 monitor window titled "COM3 (Arduino/Genuino Uno)" displays the following output:

Output
78
NHello World
1001110
116
4E
5

At the bottom of the monitor window, there are several status indicators and buttons: "Auto-rolagem" (checked), "Nenhum final-de-linha", "9600 velocidade" (selected), and "Deleta a saída".

Fonte: captura de tela do software Arduino IDE 1.8.4. elaborada pelo autor.

Na imagem, pode-se notar lado a lado o código executado e a forma que os dados foram recebidos, utilizando o terminal que a própria IDE do Arduino possui.

Analizando a Figura 4.3, podemos observar que o comando “Serial.print()” exibe a próxima mensagem exatamente na sequência da mensagem anterior. Por isso, as mensagens “N” e “Hello World” não apresentam

espaçamento entre elas. Outro ponto em destaque é que essas funções aceitam quaisquer tipos de argumentos, os quais são convertidos para a codificação ASCII (como ocorreu com a variável inteiro “a”).

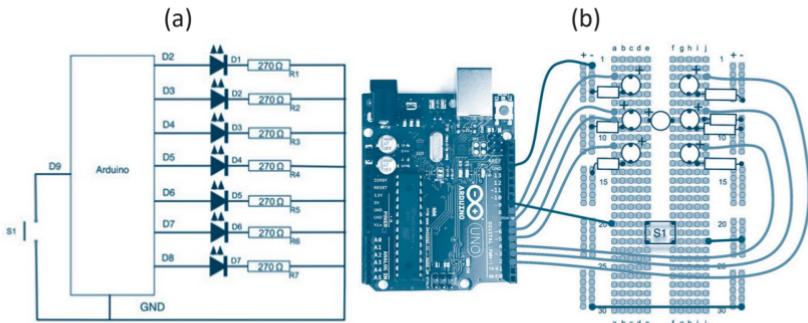
A partir do estudo dos principais periféricos, foram apresentados nesta seção como realizar a programação de um Arduino, expondo as suas principais funções, e qual é a estrutura utilizada. Para consolidar esses conhecimentos é fundamental que se pratique, por isso não deixe de testar os códigos apresentados nos exemplos a fim de compreender a fundo o funcionamento de cada um deles.

Sem medo de errar

Como desenvolvedor de sistemas embarcados, você deve projetar, montar e testar um dado eletrônico para ser utilizado em um brinquedo. O número deve ser exibido graficamente com uma disposição igual ao dado, além de ser sorteado aleatoriamente a partir do apertar de um botão. Para o desenvolvimento ágil do projeto, foi definido que ele será feito por meio da plataforma Arduino. Como o sistema é simples e não requer um alto poder de processamento, será utilizado o Arduino Uno, em uma montagem rápida, utilizando uma *protoboard*.

Para a montagem física do sistema, além do Arduino Uno, serão necessários sete resistores de $270\ \Omega$, sete LEDs de qualquer cor (3 mm ou 5 mm) e um *push-button* normalmente aberto. Os LEDs serão ligados em série com os resistores, nos pinos de 2 a 8 do Arduino, como mostra o esquemático da Figura 4.4 (a). A outra extremidade do arranjo fica ligada à referência do Arduino (GND). Para garantir a exibição gráfica de um dado, os LEDs foram arranjados em uma *protoboard* formando um H, tal como apresentado pela Figura 4.4 (b). O *push-button* foi ligado entre o pino 9 do Arduino e o GND.

Figura 4.4 | Esquemático (a) e montagem em protoboard (b) do dado eletrônico



Fonte: Monk (2014, p. 65).

Para o funcionamento do circuito é necessário que os pinos de 2 a 8 do Arduino sejam configurados como saídas digitais. Portanto, eles serão ligados ou desligados conforme o número sorteado. O pino em que o *push-button* está ligado (9) deve ser configurado como uma entrada e pelo fato de não haver um resistor de *pull-up* externo para garantir estabilidade no nível lógico, o *pull-up* interno do Arduino deve ser ativado para esse pino. Assim, quando o botão não estiver pressionado, o resistor de *pull-up* garante que o nível lógico lido seja alto. No momento em que o botão for apertado, o GND é ligado ao pino, garantindo um nível lógico baixo. Perceba que temos uma lógica invertida para o acionamento do botão.

Para garantir um efeito mais próximo ao do rolar de um dado, é interessante que o número exibido vá mudando com o passar do tempo, com uma redução na velocidade das mudanças até o ponto que ele pare em um número, mantendo esse número até o um novo apertar do botão. Para isso, pode ser utilizado um *loop* por meio de um *for* e da função *delay*. Com o objetivo de deixar o sistema ainda mais lúdico, é possível fazer que após a chegada ao número final do sorteio, ele pisque por algumas vezes, indicando o fim do sorteio.

O código que implementa o sistema do dado eletrônico está apresentado a seguir:

```
1 int ledPins[7] = { 2, 3, 4, 5, 7, 8, 6 }; // Define os pinos dos LEDs
2
3 int dicePatterns[7][7] = {
4     {0, 0, 0, 0, 0, 0, 1}, // 1
5     {0, 0, 1, 1, 0, 0, 0}, // 2
6     {0, 0, 1, 1, 0, 0, 1}, // 3
7     {1, 0, 1, 1, 0, 1, 0}, // 4
8     {1, 0, 1, 1, 0, 1, 1}, // 5
9     {1, 1, 1, 1, 1, 1, 0}, // 6
10    {0, 0, 0, 0, 0, 0, 0} }; // Apagado
11
12 int switchPin = 9; // Pino do botão
13 int blank = 6; // Posição do vetor para LEDs apagados
14
```

```

15     for (int i = 0; i < 7; i++)           // Inicializa os LEDs
16     {
17         pinMode(ledPins[i], OUTPUT);      // Como saída
18         digitalWrite(ledPins[i], LOW);   // Apagado
19     }
20     pinMode(switchPin, INPUT_PULLUP);    // Pino do botão como entrada com pull-up
21     randomSeed(analogRead(0));          // Função para alimentar o gerador de
22 }                                     // números aleatórios
23
24 void loop()
25 {
26     if (digitalRead(switchPin))        // Verifica se o botão foi apertado
27     {
28         rollTheDice();               // Chama a função que joga o dado
29     }
30     delay(100);                     // Evita o travamento do programa na
31 }                                     // verificação do botão
32
33 void rollTheDice()                  // Função do lançamento do dado
34 {
35     int result = 0;
36     int lengthOfRoll = random(15, 25); // Recebe a duração do lançamento (entre
37                                         // 15 e 25 iterações)
38     for (int i = 0; i < lengthOfRoll; i++) // Repete o sorteio conforme a duração
39     {
40         result = random(0, 6);          // Faz o sorteio de um número entre 0 e 5
41         show(result);                // Mostra o resultado nos LEDs
42         delay(50 + i * 10);           // Tempo de espera que aumenta a cada

```

```

43     }
44     //      iteração
45
46     for (int j = 0; j < 3; j++)
47         // Implementa a piscar do resultado final
48
49     {
50         show(blank);
51         // Apaga os LEDs
52
53         delay(500);
54
55         show(result);
56         // Acende os LEDs com o resultado do
57         // sorteio
58
59     }
60 }
```

Fonte: adaptado de Monk (2014, p. 66-67).



Para melhor visualizar esse código, acesse o QR Code ou o link: https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2020-2/sistemas_digitais_microprocessadores/u4/lkls202_u4s1_sis_dig_mic_QRCODE1.pdf

Com o intuito de se obter a aleatoriedade dos resultados, é utilizada a função “random”, que é uma função padrão do Arduino. Ela tem o papel de fazer um sorteio aleatório de valores na faixa especificada no chamado da função. Repare que no início do programa é chamada a função “randomSeed”, que alimenta o gerador aleatório com um valor lido pelo conversor A/D de um pino que está flutuando, ou seja, um valor desconhecido. Isso garante que os valores sorteados pela função “random” serão diferentes a cada inicialização do microcontrolador.

Enviando o programa apresentado para o Arduino, o funcionamento desejado é obtido, concluindo o projeto do sistema. Como se trata de um protótipo, o próximo passo é transformá-lo em um produto, criando uma placa de circuito impresso e uma caixa para alocar o sistema.

Avançando na prática

Projeto de um semáforo

Você faz parte da equipe de desenvolvedores de uma empresa de tecnologia, que foi contratada para a elaboração de um semáforo simples na forma de um protótipo. O objetivo é montar o sistema o mais rápido possível, em menos de uma hora, utilizando um microcontrolador em uma plataforma de prototipagem rápida. Como desenvolver esse sistema de forma rápida e funcional?

Resolução da situação-problema

Para a solução desse problema, a melhor opção é utilizar um Arduino Uno, que permite o desenvolvimento de sistemas embarcados rapidamente. Assim, para a montagem desse sistema, além do Arduino, são necessários três LEDs, um vermelho, um amarelo e um verde, além de três resistores. Cada um dos LEDs é ligado em série com um resistor e em pino do Arduino.

Como o semáforo funciona de forma sequencial, a implementação pode ser feita como a de uma máquina de estados, na qual existem três estados possíveis, um para cada estado do semáforo. A temporização de cada estado deve ser diferente, 5 segundos para o vermelho, 1 segundo para o amarelo e 10 segundos para o verde. Portanto, é necessária uma lógica que implemente essa variação. O código que pode ser utilizado para isso está apresentado a seguir:

```
1 #define PinoVermelho 2
2 #define PinoAmarelo 3
3 #define PinoVerde 4
4
5 unsigned char estado = 0;
6 unsigned char tempo;
```

```
7
8 void setup()
9 {
10     pinMode(PinoVermelho, OUTPUT);
11     pinMode(PinoAmarelo, OUTPUT);
12     pinMode(PinoVerde, OUTPUT);
13 }
14
15 void loop ()
16 {
17     if (estado == 0)
18     {
19         AcionaLeds (HIGH,LOW,LOW);
20         tempo = 5;
21         estado = 1;
22     }
23
24     else if (estado == 1)
25     {
26         AcionaLeds (LOW,LOW,HIGH);
27         tempo = 10;
28         estado = 2;
29     }
30     else if (estado == 2)
31     {
32         AcionaLeds (LOW,HIGH,LOW);
33         tempo = 1;
34         estado = 0;
```

```
35     }
36     delay (tempo*1000);
37 }
38
39 void AcionaLeds (unsigned char vm,
40                   unsigned char am
41                               unsigned char vd)
42 {
43     digitalWrite(PinoVermelho, vm);
44     digitalWrite(PinoAmarelo, am);
45     digitalWrite(PinoVerde, vd);
46 }
```

Fonte: elaborado pelo autor.

Fazendo a compilação do código e seu envio para o Arduino, o funcionamento do semáforo é obtido e, com isso, o projeto é concluído rapidamente.

Faça valer a pena

1. A comunicação serial UART é muito utilizada devido a sua facilidade de implementação. No Arduino existem bibliotecas que facilitam ainda mais essa tarefa.

Considere o seguinte comando executado após a configuração da porta serial:
“Serial.println (35, BIN);”

Assinale a alternativa que indica o valor a ser impresso na tela após a execução da linha de comando apresentada.

- a. 11011100.
- b. 00100011.
- c. 23.
- d. 43.
- e. 35.

2. Considere o seguinte código de exemplo do Arduino:

```
1 void setup() {  
2     pinMode(LED_BUILTIN, OUTPUT);  
3 }  
4  
5 void loop() {  
6     digitalWrite(LED_BUILTIN, HIGH);  
7     delay(1000);  
8     digitalWrite(LED_BUILTIN, LOW);  
9     delay(1000);  
10 }
```

Fonte: Arduino ([s.d., s.p.]).

Sabendo que LED_BUILTIN é uma definição do pino 13 do Arduino, ao qual está ligado um LED, assinale a alternativa que indica aquilo que o programa irá realizar.

- a. Envia um comando na serial a cada 1000 microssegundos, com o texto “LED_BUILTIN”.
- b. Faz a amostragem de um sinal analógico ligado no pino 13.
- c. Se trata de um código que pisca um LED, mudando o seu estado a cada 1000 microssegundos.
- d. Irá fazer com que o LED mude seu estado, entre ligado e desligado, a cada 1 segundo.
- e. Gera um sinal PWM variável entre HIGH e LOW a cada 1000 microssegundos.

3. Existem diversos protocolos de comunicação que um microcontrolador pode utilizar para se comunicar com os mais diversos tipos de dispositivos. O protocolo USART é muito utilizado para a comunicação do microcontrolador com outros dispositivos, como um computador por exemplo.

Acerca do protocolo USART, assinale a alternativa correta.

- a. Esse protocolo permite a comunicação entre mais de dois dispositivos.
- b. O modo de operação assíncrono permite um regime de comunicação *full-duplex*, em que ambos os dispositivos podem transmitir e receber dados ao mesmo tempo.
- c. No modo de operação assíncrono, utilizam-se três fios para a comunicação, mais o terra, sendo eles: *clock*, transmissor e receptor.
- d. No modo de operação síncrono, utiliza-se um par de fios, mais o terra, sendo um para transmissão e outro para recepção.
- e. O *baud rate* da comunicação corresponde à paridade utilizada para a verificação dos dados, podendo ser par ou ímpar.

Seção 2

Novas arquiteturas e tendências

Diálogo aberto

Prezado aluno,

Os avanços que a computação teve nos últimos anos permitiu o desenvolvimento de sistemas cada vez mais inteligentes e mais presentes no nosso dia a dia. Uma das vertentes que mais contribuiu para isso foi a inteligência artificial.

A aplicação da inteligência artificial permite a detecção facial para sistemas de segurança, sejam eles para o controle de acesso em prédios e empresas, sejam para o simples desbloqueio de nossos smartphones. Permite também a otimização dos sistemas de previsão, como os utilizados em meteorologia, e de reconhecimento de imagens e padrões, com o *Machine Learning*. Todos esses exemplos mostram como a inteligência artificial é um assunto relevante para quem trabalha com tecnologia.

Falando ainda sobre os avanços tecnológicos, agora relacionados ao hardware dos dispositivos, uma arquitetura que ganhou destaque é a ARM (*Advanced RISC Machine*). Ela está presente na maioria dos sistemas embarcados e domina o mercado dos processadores de smartphones. Dessa forma, conhecer a fundo essa arquitetura e como realizar a programação de um microcontrolador desse tipo é relevante para o profissional que desenvolve sistemas digitais.

Retomando o contexto em que você trabalha no desenvolvimento de sistemas embarcados, você foi incumbido de desenvolver um novo projeto com a utilização de microcontrolador. A sua empresa foi contratada para desenvolver um sistema de aquisição de dados para ser colocado em uma estação meteorológica e você será responsável por programar a comunicação do microcontrolador com uma memória externa, que será utilizada para armazenar os dados coletados temporariamente.

A memória externa é do tipo *flash*, isto é, possui uma velocidade de acesso alta e os dados continuam armazenados mesmo com o seu desligamento, com interface de comunicação SPI. Como o sistema requer o processamento de uma grande quantidade de dados, optou-se por utilizar um microcontrolador ARM de 32 bits. Assim, você deve realizar a programação desse dispositivo para que ele realize operações de leitura e escrita nessa memória externa, permitindo que o código seja incrementado com outras funções posteriormente. Como configurar o periférico de

comunicação do ARM? Como implementar o acesso à memória? Como elaborar a lógica do programa?

Para responder a essas perguntas sobre a arquitetura ARM você será levado a estudar suas características a fim de compreender o porquê ela é tão utilizada atualmente. Somado a isso, você irá estudar os conceitos básicos da inteligência artificial, suas vertentes e formas de implementação.

Vamos iniciar os estudos?

Não pode faltar

Quando se trabalha com sistemas digitais, conhecer as tecnologias emergentes é algo muito importante. Assim, além de saber como programar um microcontrolador, saber aplicar novas tecnologias na programação torna o profissional ainda mais capacitado. Uma dessas tecnologias emergentes é a **inteligência artificial** (IA).

A criação da inteligência artificial permitiu que diversas coisas que utilizamos em nosso cotidiano se tornasse possíveis, como realizar buscas na internet, acionar um eletrodoméstico, conversar por meio do celular e controlar uma fábrica automatizada. Para que isso se tornasse possível, fez-se necessário o desenvolvimento de diversos algoritmos e dispositivos criados para simular o pensamento ou a ação humana com o objetivo de facilitar operações específicas (LUGER, 2013).

Com o advento da IA, a tecnologia passou a contar com a simulação de processos que auxiliam no reconhecimento de padrões, em execução de tarefas repetitivas e em tomada de decisões. Assim, fica claro que existem diversas áreas de estudo dentro da IA, com diferentes modos e abordagens dos aspectos da inteligência. Portanto, existe uma série de elementos que se manifestam de diferentes formas e que correlacionam de maneiras distintas os processos de IA com os mecanismos do cérebro e da mente humana (MEDEIROS, 2018).

A IA é categorizada em dois tipos de processos: o de pensamento, que está relacionado aos mecanismos de raciocínio, e o de ação ou comportamento, relativo ao comportamento do artefato. De forma transversal a esses processos, é considerada a similaridade em relação ao ser humano ou a racionalidade envolvida. Com isso, as definições de IA são classificadas em um framework de quatro categorias (MEDEIROS, 2018):

- Sistemas que pensam como os seres humanos: está relacionada à automatização de atividades referentes ao pensamento humano, como a tomada de decisões e a resolução de problemas. Um exemplo desse

tipo de sistema é um software inteligente capaz de tomar decisões com base em conhecimentos adquiridos por um ser humano especialista.

- Sistemas que pensam racionalmente: são sistemas computacionais que tornam possível perceber, raciocinar e agir. Como exemplo, pode-se citar um sistema que execute raciocínios conforme regras de lógica e um sistema de jogo de xadrez que execute regras predefinidas.
- Sistemas que atuam como seres humanos: esses sistemas executam funções que exigem inteligência mesmo quando executadas por humanos. Um exemplo é um robô antropomórfico que executa movimentos semelhantes aos do ser humano.
- Sistemas que atuam racionalmente: estão relacionados ao projeto de agentes inteligentes, como robôs que executam alguma atividade no ambiente de forma diferente da humana. Um exemplo disso é um robô aspirador de pó, que executa um algoritmo de limpeza desviando de obstáculos baseado na detecção feita por sensores de proximidade. Outro exemplo é um robô que opera realizando solda em uma fábrica.

Definidas as categorias e analisando os exemplos citados, fica evidente que a IA não se trata somente de um software, mas está relacionada também ao hardware de um sistema. Aqui fica evidente o papel de um microcontrolador nesse processo, que é justamente um elemento integrador de hardware e software, já que ele pode ser programado e é capaz de ler sinais e acionar dispositivos.

A pesquisa e o desenvolvimento da IA é resultado da convergência de diversas áreas de conhecimento com abordagens variadas, que criaram algumas linhas fundamentais de pesquisa. Essas linhas culminaram em diferentes tecnologias utilizadas dentro da IA, as quais são aplicadas para a solução de diferentes tipos de problemas.

A primeira linha de pesquisa da IA é chamada de **conexionista**; é baseada no cérebro e tem como objetivo construir cérebros artificiais, com neurônios artificiais que tenham a capacidade de simular a maneira como os mecanismos eletroquímicos dos cérebros biológicos funcionam. Assim, esse tipo de sistema é voltado ao desenvolvimento da arquitetura de dispositivos que simula a interação das células biológicas para que ocorram os processos relacionados à inteligência. Podem ser citados dois exemplos de sistemas pautados nessa prerrogativa (MEDEIROS, 2018):

- Redes neurais artificiais (RNA): são uma linha de pesquisa que tem como característica o desenvolvimento de um sistema para o reconhecimento de padrões, previsão e tomada de decisões baseadas no uso

de redes de unidades conectadas e treinadas por meio de algoritmos que funcionam com base em amostras do mundo real.

- Sistemas imunológicos artificiais: utilizam como premissa o funcionamento do sistema imunológico dos seres vivos, que é capaz de reconhecer um elemento estranho e desenvolver uma reação defensiva, que resulta na produção de anticorpos para eliminar os抗ígenos invasores.

A outra linha de pesquisa da IA é denominada de **simbólica**, pois tem como objetivo trabalhar com processos inteligentes utilizando linguagens baseadas em lógica e na construção de redes para solucionar problemas e simular o conhecimento de um especialista de determinado contexto. Portanto, essa linha tem como princípio a mente, atuando no processamento de símbolos de forma semelhante ao que ocorre na mente humana. Alguns exemplos de sistemas simbólicos de IA são (MEDEIROS, 2018):

- Sistemas baseados em conhecimento: são linguagens, como a *LISP*, que trabalha com a representação de conhecimento em forma de lista, e a *Prolog*, que permite a manipulação de símbolos por meio da representação de conhecimento no formato de fatos e regras.
- Sistemas especialistas: são sistemas em que o conhecimento de um humano especialista em determinada área é representado por uma linguagem, permitindo o diagnóstico de situações e execução de ações como um humano faria.
- Sistemas ontológicos: representam o conhecimento obtido por consenso em áreas específicas do conhecimento humano.

Uma terceira vertente de estudo da IA é a **evolucionária**, que está relacionada ao modo como se processa a evolução biológica sobre o planeta, buscando simular processos evolucionários semelhantes em sistemas computacionais visando a resolução de problemas. Uma área dessa vertente é a dos **algoritmos genéticos**, que buscam implementar conceitualmente uma solução inicial, a qual evolui ao longo da execução do próprio algoritmo (MEDEIROS, 2018).

Como se pode perceber, a IA possui três áreas de pesquisa: a simbólica, a conexionista e a evolucionária. Todas elas podem se traduzir em algoritmos, os quais podem ser executados em um computador ou em um sistema embarcado composto por hardware e software. No caso de um sistema embarcado, é necessário optar pela utilização de um microprocessador que seja capaz de executar um algoritmo de IA, o qual requer um poder de processamento

relativamente alto. Uma possível opção para isso são os microcontroladores e microprocessadores com arquitetura ARM.

O acrônimo **ARM** significa *Advanced RISC Machine* e trata-se de uma empresa fundada em 1980 na Inglaterra, que se caracteriza pelo desenvolvimento de microprocessadores. Na verdade, a empresa é responsável pelo desenvolvimento dos chips e é detentora dos direitos sobre a arquitetura, limitando-se a licenciar os projetos a preços modestos para outros fabricantes, que podem optar por diversos tipos de licença, incluindo opções que permitem modificar chips e incluir componentes adicionais. Portanto, o ARM não se trata especificamente de um microcontrolador, mas sim de uma arquitetura de núcleo presente em microcontroladores e microprocessadores de diversos fabricantes.

A arquitetura ARM é utilizada para fabricação de processadores de 32 bits cuja principal aplicação é o desenvolvimento de sistemas embarcados. Esses processadores visam à simplificação das instruções para obter a máxima eficiência por ciclo, consequentemente apresentam tarefas menores e com ciclos mais curtos. Seu projeto foi concebido buscando o melhor desempenho possível, com a limitação de ser simples, ocupar pouca área e ter baixo consumo de energia.

Estima-se que em 2015 foram fabricados mais de 14,9 bilhões de dispositivos, enquanto, em 2016, essa marca chegou a 16,7 bilhões. Segundo a empresa, as tecnologias ARM atingem cerca de 80% da população mundial, estando presente em PDAs (*Personal Digital Assistant*, em português Assistente Pessoal Digital), telefones celulares, calculadoras, periféricos de computadores e aplicações industriais. Atualmente, mais de 45 milhões de chips ARM são vendidos por fabricantes licenciados, dentre os quais se destacam: STMicroelectronics, NXP, Texas Instruments, entre outros (GARCIA, 2017).

Como o próprio nome diz, os microcontroladores ARM apresentam um conjunto de instruções RISC (*Reduced Instruction Set Computer*), fabricados com uma quantidade menor de transistores em relação às máquinas CISC (*Complex Instruction Set Computer*). Essa abordagem permite uma redução de custos, a liberação de calor e o consumo de energia. Além disso, a criação de chips com múltiplos núcleos de processamento é mais simples. As instruções têm tamanho fixo de 32 bits, alinhadas em 4 bytes consecutivos da memória.

Os processadores ARM podem ser divididos em duas grandes famílias. A primeira é a dos chips mais antigos (ARM7, ARM9 e ARM11), que ainda são os mais usados, e a segunda é a linha mais atual, chamada de *Cortex*.

Os processadores *Cortex* são subdivididos em Linha A, Linha R e Linha M de acordo com suas aplicações. A Linha A é destinada a processadores de alto desempenho, geralmente usados em aplicações mobile e plataformas, por exemplo smartphones. A Linha R é caracterizada em aplicações de tempo real, como em sistemas embarcados de segmentos automotivos e de controle industrial. Por fim, a linha M, representa o uso de processadores na construção de microcontroladores dedicados a aplicações de baixo custo e baixo consumo de energia. Também são utilizados na construção de dispositivos de IoT (internet das coisas) devido a suas características de segurança (GARCIA, 2017).

Analizando a linha de processadores utilizados em microcontroladores, Linha M, três famílias se destacam: Cortex-M0, Cortex-M3 e Cortex-M4.

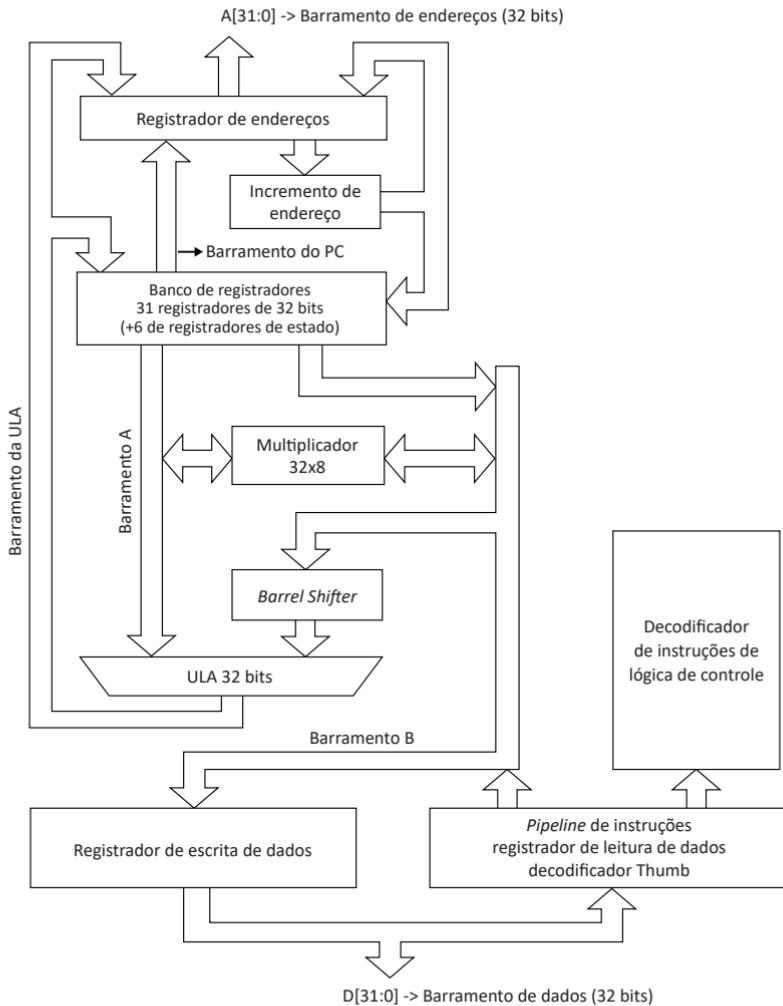
- Cortex-M3: é o processador líder da indústria de 32 bits para aplicações em tempo real. Introduziu recursos como controle de prioridade de interrupções e Unidade de Proteção de Memória (MPU – *Memory Protection Unit*). Apresenta arquitetura ARMv7-M do tipo Harvard.
- Cortex-M4: conceitualmente o Cortex-M4 é um Cortex-M3 acrescido de instruções de processamento digital de sinais (DSP – *Digital Signal Processor*) e apresenta uma unidade opcional de ponto flutuante (formato de representação digital de números racionais mais precisos e que facilita a programação). Sua arquitetura é a ARMv7E-M do tipo Harvard.
- Cortex-M0: são os processadores ARM de 32 bits que apresentam a maior eficiência em relação ao consumo de energia. Possuem arquitetura ARMv6-M do tipo Von Neumann e um conjunto de instruções RISC com 56 instruções. Foram desenvolvidos para oferecer performance de processadores de 32 bits com um custo de processadores de 8 bits. Essa família também apresenta a opção de uma Unidade de Proteção de Memória. Essa arquitetura apresenta a novidade de operar com dois estágios de *pipeline*. Assim, o processador acessa menos a memória e apresenta um consumo de energia menor se comparado às arquiteturas com três estágios de *pipeline*, como acontece com os núcleos Cortex-M3 e M4.

Para que possamos saber como programar um ARM, é importante conhecer sua arquitetura. Assim, vamos tomar como base uma CPU (*Central Processing Unit*) ARM7, que pode estar presente tanto em um microcontrolador quanto em um microprocessador para smartphone. Essa CPU, pautada em uma máquina RISC de 32 bits, tem as seguintes características:

- Estão disponíveis sete modos de operação da CPU, incluindo o modo normal de usuário, dois modos para o processamento de interrupções, dois modos para a utilização em tarefas e sistemas operacionais e dois modos para o processamento de outras exceções.
- Existem trinta e sete registradores internos, sendo trinta e um de uso geral e seis de estado da CPU.
- Presença da arquitetura *Load-and-Store*, na qual qualquer operação de dados é feita somente entre registradores da CPU. As instruções especiais de carga (*load*) e armazenamento (*store*) são utilizadas para a troca de dados entre a memória e os registradores da CPU.
- Instruções condicionadas a um estado predefinido das *flags* de estado da CPU, nas quais uma instrução é executada somente se a condição especificada nela for a mesma que a indicada pelas *flags* da CPU. Caso isso não ocorra, a instrução não é executada e não produz efeito.
- Utilização do modelo Von Neumann, havendo somente um espaço de endereçamento e um único conjunto de barramentos de controle/endereços/dados, que é compartilhado pelos registradores de controle dos periféricos, memória RAM e *flash*. Além disso, não há diferenciação entre áreas de memória voltadas para o programa ou dados. Portanto, a CPU pode fazer a busca de instruções e leitura de dados tanto na memória *flash* quanto na RAM.
- Utilização de pipeline de três níveis, permitindo, ao mesmo tempo, a execução de uma instrução, a decodificação de outra e a busca na memória de uma terceira.

A fim de compreender melhor as características apresentadas, é importante conhecer a estrutura de uma CPU ARM7. Para isso, observe a Figura 4.5, que mostra um diagrama simplificado dos componentes dessa CPU e como eles estão arranjados. Perceba que a ULA (Unidade Lógica Aritmética) possui duas entradas (barramentos A e B), que podem alimentá-la com operados vindos do banco de registradores da CPU. Adicionalmente os dados que chegam à ULA pelo barramento B podem ser deslocados de 0 a 31 bits, em um deslocamento bit a bit, para a direita ou esquerda, pelo *Barrel Shifter*. Também é possível identificar a presença do Multiplicador 32x8, capaz de realizar operações de multiplicação de forma independente da ULA e de operações de multiplicação e acúmulo em conjunto com ela (PEREIRA, 2017).

Figura 4.5 | Diagrama de blocos simplificado da CPU ARM7



Fonte: Pereira (2017, p. 66).

Assimile

Um *barrel shifter* nada mais é do que um deslocador de bits capaz de realizar o deslocamento de múltiplos bits à esquerda ou à direita em uma única operação. A grande diferença entre esse circuito e os demais registradores de deslocamento é que ele é montado somente a partir da lógica combinacional, sem a utilização de nenhum *flip-flop* ou sinal

de *clock*. Assim, para a montagem de um *barrel shifter*, são utilizados multiplexadores e seletores de dados.

Para realizar a programação de um microcontrolador, que não faça parte da plataforma de desenvolvimento Arduino, é necessário selecionar uma IDE (do inglês, *Integrated Development Environment*, ou Ambiente de Desenvolvimento Integrado) e um compilador que sejam compatíveis com o microcontrolador selecionado. Para a programação de um microcontrolador ARM isso não é diferente. Além disso, a estrutura do programa é diferente.

Exemplificando

Para a programação de microcontroladores ARM fabricados pela STMicroelectronics, uma opção é utilizar o STM32CubeIDE, disponibilizado gratuitamente pela própria fabricante. Essa IDE possui já integrado o compilador (*GNU Compiler Collection*), que consiste em uma coleção de compiladores para múltiplas plataformas e que é utilizado para a programação de um ARM.

A estrutura de um programa desenvolvido em C é composta por um arquivo cujo início contém comandos para a inclusão de arquivos externos, depois a declaração de variáveis globais, protótipos de funções e funções, culminando na função principal (*main*), que é obrigatória para todos os programas. No caso de um programa muito extenso, a recomendação é a criação de funções que podem ser colocadas em outros arquivos, criando assim uma biblioteca, incluída no arquivo principal. Deve ficar claro que todo o programa deve possuir somente uma função principal (*main*).

Durante a programação, também é importante definir os tipos de variáveis a serem utilizadas, as quais possuem diferentes tamanhos e têm a função de armazenar dados. Além do tipo, deve-se optar pela utilização de variáveis locais, disponíveis somente dentro da função para a qual foram declaradas, ou globais, que podem ser utilizadas por qualquer função do programa. Todas as variáveis podem ser inicializadas ou não com um valor.

Após essa breve revisão de como é estruturado um programa em C, você deve estar se perguntando como é realizada efetivamente a programação de um ARM. A lógica utilizada não se difere da aplicada em um Arduino de 8 bits, a maior diferença fica por conta da maior complexidade na configuração do microcontrolador e seus periféricos.

Isso ocorre, pois a quantidade de funcionalidades é maior e culmina em uma maior quantidade de registradores para serem configurados. Porém, isso é facilitado pela utilização de bibliotecas disponíveis na grande maioria das IDEs.

Para ilustrar como acontece a configuração de um microcontrolador ARM, vamos estudar dois protocolos de comunicação digital utilizados para o “interfaceamento” de dispositivos externos, como sensores e memórias. Esses protocolos são o SPI e o I²C, que ainda não foram tratados.

O periférico **SPI** é um protocolo de comunicação de dados seriais síncronos utilizados pelos microcontroladores para se comunicarem com outros componentes, formando uma rede. Ele estabelece o conceito de mestre e escravo, no qual um único dispositivo mestre deve assumir o controle da comunicação, gerando o sinal de sincronismo (*clock*).

Como seu funcionamento é firmado em um registrador de deslocamento, a comunicação *full-duplex* torna-se uma exigência, ou seja, para cada bit enviado pelo transmissor, um bit também deve ser recebido. Sendo assim, os sinais de comunicação apresentam direção fixa, com barramentos independentes para transmissão e recepção. O controle do barramento para estabelecer nível lógico alto ou nível lógico baixo é definido por um pino ligado a dois transistores na configuração *push-pull*.

Assimile

A configuração *push-pull* é composta por dois elementos ativos (transistores, por exemplo) e tem a função de aplicar um nível lógico preciso na saída do arranjo com uma alta velocidade de comutação.

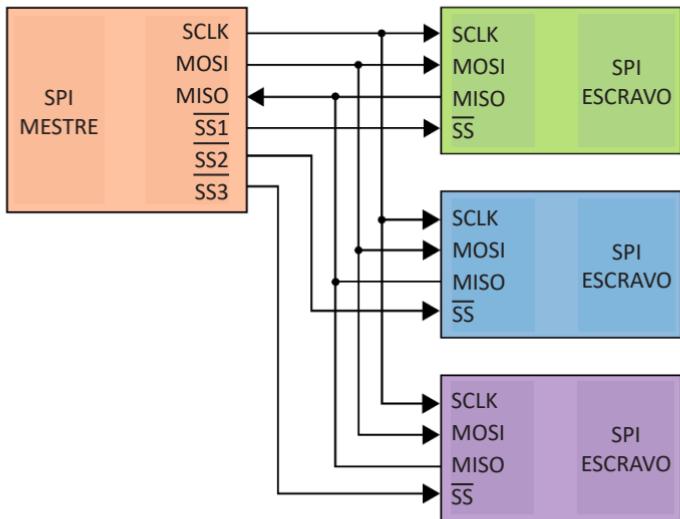
Esse protocolo apresenta uma taxa máxima de transferência de dados que pode chegar a mais de 50 Mbps, o que é muito superior à taxa de transferência apresentada pela comunicação UART (*Universal Asynchronous Receiver and Transmitter*). Outra vantagem é que a comunicação pode apresentar mais de dois dispositivos, desde que o sistema tenha apenas um mestre. Assim, cada dispositivo escravo tem um terminal de seleção para habilitá-lo. Portanto, o número de fios no barramento de um sistema de comunicação SPI com “n” escravos é de 3 + n. No Quadro 4.1, estão definidos os pinos utilizados para essa comunicação e a Figura 4.6 mostra como são feitas as ligações físicas nesse protocolo.

Quadro 4.1 | Descrição dos pinos de comunicação SPI

Pino	Nome padrão	Significado	Nomes alternativos
Do mestre para o escravo	MOSI	<i>Master Output Slave Input</i>	SDO, DO, SO
Do escravo para o mestre	MISO	<i>Master Input Slave Output</i>	SDI, DI, SI
<i>Clock</i>	SCLK	<i>Serial Clock</i>	SCK, CLK
Seleção do escravo	SS	<i>Slave select</i>	CS, nSS, nCS

Fonte: elaborado pelo autor.

Figura 4.6 | Conexão de dispositivos por SPI



Fonte: adaptada de Wikimedia Commons.

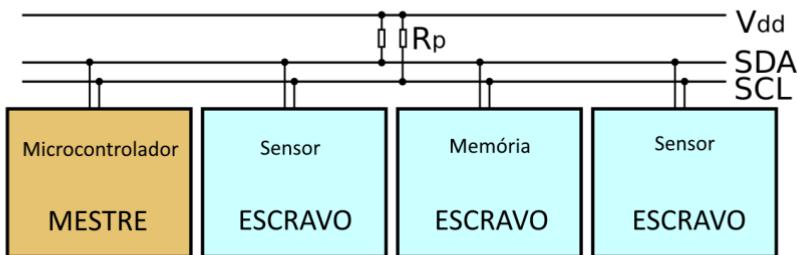
A desvantagem desse periférico é o número elevado de barramentos quando são utilizados vários escravos. Além disso, sua utilização está associada a distâncias curtas, normalmente menores que 30 centímetros.

Dentro de um microcontrolador, o periférico pode ser configurado como mestre ou escravo, porém é mais comum a operação no modo mestre, no qual ele controla os dispositivos ligados a ele. Assim, considerando um microcontrolador ARM7, devem ser feitas as seguintes configurações para

o funcionamento do periférico, que podem ser realizadas diretamente nos registradores ou por meio de bibliotecas: configuração da velocidade de comunicação; configuração dos pinos que estarão multiplexados com SPI para o modo alternativo de saída *push-pull*; habilitar a interface SPI; selecionar o modo de operação (mestre ou escravo); selecionar o modo da interface SPI, com *clock* ativo em nível alto ou baixo e dado armazenado na borda de subida ou descida; configuração do tamanho do caractere (8 ou 16 bits); configuração do tamanho dos buffers de transmissão e recepção; limpeza do buffer de recepção. Repare que são diversas configurações que devem ser realizadas, assim, é importante não esquecer de realizar nenhuma delas para garantir a operação correta do periférico.

O protocolo I²C é um barramento serial *multimestre* desenvolvido pela Philips para conectar vários periféricos em um sistema embarcado. Ele utiliza duas linhas de barramento bidirecionais para conectar até 128 dispositivos diferentes, sendo um barramento para o sinal de *clock* (chamado de SCL) e outro para dados (chamado de SDA), conforme é demonstrado na Figura 4.7. Consequentemente, esse periférico só pode operar em sistema *half-duplex*, no qual mestre e escravo enviam e recebem dados, porém nunca ao mesmo tempo. Nesse protocolo, cada dispositivo possui um endereço diferente, que seleciona o dispositivo no momento do envio dos dados.

Figura 4.7 | Conexão de dispositivos por I²C



Fonte: adaptada de Wikimedia Commons .

A taxa máxima de transferência de dados desse periférico pode chegar a até 1 Mbps. A comunicação se inicia quando o barramento SDA vai para nível lógico baixo enquanto o barramento SCL estiver em alto e termina quando ocorrer uma transição de baixo para alto do barramento SDA enquanto o sinal de SCL está em alto.

Sua implementação em hardware é feita conectando-se um resistor de *pull-up* em cada barramento, pois o controle do nível lógico do barramento é feito por um transistor na configuração dreno-aberto. Assim, quem garante o nível alto nesse barramento é esse resistor.

De forma semelhante à interface SPI, a interface I²C também necessita de configuração. Em um ARM7, as configurações que devem ser realizadas são as seguintes: habilitação da interface; se será gerada ou não uma condição de *start* na linha; se será gerado um bit de reconhecimento (*acknowledge*); se será gerado um bit de parada; configuração da velocidade de operação da interface; e dependendo do microcontrolador, configuração dos pinos que estarão multiplexados com I²C para o modo alternativo.

Refita

Realizar a configuração e utilização de um microcontrolador ARM com bibliotecas facilita muito a programação. Existiria então, alguma vantagem em se utilizar o acesso direto aos registradores na programação?

A partir dos conceitos aqui apresentados, você pôde compreender como é estruturada a inteligência artificial e quais são as linhas de pesquisa que ela possui. Somado a isso, você estudou a arquitetura ARM, compreendendo como realizar a configuração de dois periféricos de comunicação, o I²C e SPI. Mas os estudos não terminam por aqui, vamos continuar trabalhando com microcontroladores ARM na próxima seção.

Até lá!

Sem medo de errar

Você trabalha no desenvolvimento de sistemas embarcados e foi incumbido de desenvolver um novo projeto que utiliza microcontrolador ARM. Nesse projeto, você deve realizar a programação do microcontrolador para que ele realize operações de leitura e escrita em uma memória *flash*.

Para o desenvolvimento desse projeto, será utilizado o microcontrolador STR711, fabricado pela STMicroelectronics, e a memória M25P10A, fabricada pela *Micron*. O microcontrolador possui uma CPU ARM7 de 32 bits, duas interfaces de comunicação I²C, sendo uma multiplexada com SPI e uma porta USB (STMICROELECTRONICS, 2013). A memória possui capacidade de 1 Mbit arranjado em 125 mil endereços, que armazenam um byte cada, e possui comunicação SPI (MICRON, 2014).

A memória em questão possui oito pinos, sendo três para comunicação (entrada de dados, saída de dados e *clock*), um para a seleção do chip dentro do barramento SPI, um para proteção contra escrita e outro para interromper a comunicação, além dos pinos de alimentação. Para se estabelecer a comunicação com a memória, com os pinos de proteção desabilitados e

o chip selecionado, o dispositivo mestre (microcontrolador) deve enviar o comando desejado via SPI. Existem comandos para habilitar e desabilitar a escrita, ler e escrever no registrador de status, apagamento de um setor ou da memória por completo, leitura e escrita da memória.

Para realizar a leitura da memória, deve ser enviado o comando correspondente (0x03), seguido de três bytes de endereço. Então, o mestre deve enviar pulsos de *clock* para que a memória disponibilize os dados. Para a leitura do endereço seguinte, basta enviar mais oito pulsos de *clock* e assim sucessivamente. O fim da leitura acontece quando o pino de seleção da memória é desativado.

O processo de escrita na memória é bastante semelhante. Envia-se o byte do comando (0x02) seguido dos três bytes de endereço e dos dados a serem escritos (quanto bytes se desejar). Para finalizar a escrita, basta desabilitar o pino de seleção.

Sabendo o funcionamento da memória, é possível elaborar uma biblioteca para realizar as principais operações de leitura e escrita, de um ou vários bytes da memória. O código a seguir mostra como pode ser elaborada a lógica para envio e recepção de dados via SPI e uma função para a leitura da memória.

```
1 // ***** Transfere um byte pelo pino MOSI e recebe outro pelo MISO *****
2 unsigned char spi_transfere (unsigned char dado)
3 {
4     unsigned char temp;
5
6     while (BSPI0_CSR2 & bRFNE)           // Verifica se existem dados no buffer de recepção
7
8     temp = BSPI0_RXR;                  // Esvazia o buffer de recepção
9
10    while (BSPI0_CSR2 & bTFF);        // Verifica se não há dado já sendo transmitido
11
12    BSPI0_TXR = dado << 8;          // Transmite o dado
13
14    while (!(BSPI0_CSR2 & bRFNE));   // Aguarda o dado ser recebido
15
16    while(BSPI0_CSR2 & bRFNE)
17
18        temp = BSPI0_RXR>>8;      // Faz a leitura do dado recebido
19
20    return (temp);
21
22 }
23
24 // ***** Lê "num_byte" bytes da memória M25P10A *****
```

```

16 void M25P10A_read_bytes (unsigned int endereco, unsigned int num_byte,
17                           unsigned char *end_destino)
18 {
19     GPIO0PD &= ~BIT4;           // Ativa o pino de seleção (ativo em baixo)
20     spi_transfere(0x03);       // Envia o comando relativo a leitura
21     spi_transfere(endereco >> 16); // Envia o primeiro byte do endereço
22     spi_transfere(endereco >> 8); // Envia o segundo byte do endereço
23     spi_transfere(endereco);    // Envia o terceiro byte do endereço
24     for(;num_byte;num_byte--)   // Repete conforme o número de bytes a serem lidos
25         *end_destino++ = spi_transfere(DUMMY); // Faz a leitura
26     GPIO0PD |= BIT4;          // Desabilita a memória
27 }

```

Fonte: adaptado de Pereira (2017, p. 416-417).



Para melhor visualizar esse código, acesse o QR Code ou o link: https://cm-cls-content.s3.amazonaws.com/ebook/embed/qr-code/2020-2/sistemas_digitais_microprocessadores/u4/lklis202_u4s2_sis_dig_mic_QRCODE2.pdf

Perceba que é utilizada uma mesma função para a transmissão e envio de dados (`spi_transfere`). Isso ocorre pois, para a recepção de dados, devem ser gerados pulsos de *clock* e isso acontece somente com o envio de dados. Assim, um dado qualquer é enviado pelo microcontrolador para a geração do *clock*, que é utilizado pela memória para o envio de dados. Repare ainda que são utilizados comandos que acessam diretamente os registradores do microcontrolador, o que dificulta a sua programação.

Para que o microcontrolador possa se comunicar via SPI, ele deve receber a configuração de diversos registradores. O código a seguir mostra um exemplo de configuração desse periférico para o microcontrolador utilizado no projeto.

```

1 // Configuração da interface BSPI0:
2 // Buffer de transmissão e de recepção = 1 byte
3 // BSPI0 no modo mestre de 8 bits CPOL=0, CPHA=0

```

```
4 // Clock = 48 MHz / 2 / 6 = 4 MHz
5 BSPI0_CSR1 = SPI_RXFIFO_1W + SPI_8BITS + bMSTR + bBSPE;
6 BSPI0_CSR2 = SPI_TXFIFO_1W;
7 BSPI0_CLK = 6; // Velocidade da SPI = PCLK1 / 6 = 4 MHz
8 PCU_BOOTCR = bSPI0EN; // habilita a interface BSPI0
```

Fonte: Pereira (2017, p. 419).

Com a configuração do microcontrolador feita corretamente e com a biblioteca para acesso à memória funcional, basta realizar a ligação física da memória com o microcontrolador corretamente. Feito isso, chega-se à conclusão do projeto incumbido a você.

Avançando na prática

Temporizadores em um microcontrolador ARM

Considere que você trabalha como desenvolvedor de sistemas embarcados e foi incumbido de realizar a programação de um temporizador de um microcontrolador ARM da família STR71x. O objetivo é que seja gerada uma temporização precisa utilizando esse periférico e para isso deve-se desenvolver uma biblioteca com a função *delay*, que opere de forma semelhante à presente no Arduino. Em um primeiro momento você deve estudar como se configura um temporizador e como ele pode ser utilizado. Quais são as informações que serão descobertas com esse estudo?

Resolução da situação-problema

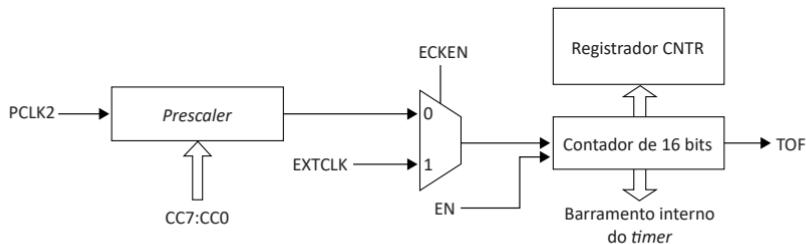
Praticamente todos os microcontroladores possuem um ou mais temporizadores (*timers*) como periférico. Eles possuem a possibilidade de realizar contagens de 8, 16 ou 32 bits, possuindo diversas funcionalidades. No caso de um microcontrolador ARM da família STR71x, estão presentes quatro *timers*. Cada um desses *timers* possui um contador de 16 bits com divisor prévio (prescaler) programável e dois canais ligados aos pinos, que podem ser utilizados para:

- Captura: na qual é possível realizar a medição do período de sinais externos.

- Comparação: aplicada na geração de sinais periódicos e comparação de sinais.
- PWM: para a geração de sinais modulados em largura de pulso, de modo que os dois canais são utilizados para a geração de uma onda PWM.
- Pulso único: o *timer* gera um pulso de largura programável, que é disparado por uma entrada externa.

O núcleo central de um dos *timers* é composto por diversos elementos que permitem o ajuste preciso da temporização a ser gerada. Nele encontram-se um divisor (*prescaler*) programável, que tem a função de dividir o sinal de *clock* interno por um fator entre 1 e 256, um multiplexador para a seleção da origem do sinal de *clock* e o contador principal, como está apresentado na Figura 4.8.

Figura 4.8 | Contador principal de um microcontrolador AMR da família STR711x



Fonte: Pereira (2017, p. 227).

Analizando a Figura 4.8, o sinal PLCK2 corresponde ao *clock* interno gerado pelos circuitos osciladores presentes no microcontrolador e CC7:CC0 são os bits que indicam qual será a divisão do *clock* feita pelo *prescaler*. O multiplexador que tem como seleção o bit ECKEN e direciona para sua saída o sinal vindo do *prescaler* ou um sinal de *clock* externo (EXTCLK). O sinal selecionado pelo multiplexador é aplicado no contador, que é habilitado pelo sinal EM. O estouro da contagem do contador gera o acionamento de uma flag (TOF) e o valor da contagem é armazenado no registrador CNTR, que pode ser lido pelo programa. Vale ressaltar que todos os bits citados estão presentes nos registradores de configuração do *timer*.

A partir do estudo da operação de um *timer* e de suas funcionalidades, você já consegue elaborar um código para configurá-lo e para criar uma função do tipo *delay*.

Faça valer a pena

1. Os protocolos de comunicação SPI e I²C são muito utilizados para a interface do microcontrolador com sensores digitais e memórias externas. Portanto, conhecer o funcionamento desses periféricos é importante para a programação de um microcontrolador.

Acerca dos protocolos SPI e I²C, assinale a alternativa correta.

- a. Tanto o protocolo SPI quanto o I2C permitem a comunicação entre apenas dois dispositivos.
- b. O funcionamento do protocolo SPI é baseado no processo *shift-register*, fazendo com que a comunicação seja *full-duplex*.
- c. O protocolo I²C possui dois canais de comunicação por onde os dados trafegam, o MISO e o MOSI.
- d. O protocolo SPI possui dois canais de comunicação por onde os dados trafegam, o SCL e o SDA.
- e. O protocolo SPI exige que sejam colocados resistores de *pull-up* em cada um dos barramentos de comunicação.

2. A inteligência artificial possui diversas características, o que gera diferentes categorias, dentro de um framework que engloba os tipos de processos e similaridades com o ser humano. Dessa forma, existe um determinado tipo de sistema dentro da inteligência artificial que está relacionado ao projeto de agentes inteligentes, como robôs que executam alguma atividade no ambiente de forma diferente da humana. Um exemplo desse tipo de sistema é um robô que opera realizando solda em uma fábrica.

Assinale a alternativa que indica a qual tipo de sistema dentro da inteligência artificial o trecho se refere.

- a. Sistemas híbridos, que pensam e atuam como os seres humanos.
- b. Sistemas que pensam como os seres humanos.
- c. Sistemas que pensam racionalmente.
- d. Sistemas que atuam racionalmente
- e. Sistemas que atuam como seres humanos.

3. A programação de um microcontrolador exige o conhecimento de sua arquitetura. Considere as seguintes afirmações sobre as características da CPU de um ARM7:

- I. Existem trinta e sete registradores internos, sendo trinta e um de uso geral e seis de estado da CPU.
- II. Qualquer operação aritmética realizada com dados é feita somente entre registradores da CPU.
- III. Existe somente um espaço de endereçamento e um único conjunto e barramentos de controle/endereços/dados, que é compartilhado pelos registradores de controle dos periféricos, memória RAM e *flash*.
- IV. Não há utilização de *pipeline* em nenhum nível.

Assinale a alternativa que indica quais afirmativas são corretas.

- a. Somente I.
- b. Somente II e III.
- c. Somente II, III e IV.
- d. Somente I, II e III.
- e. I, II, III, IV.

Seção 3

Aplicações para processadores ARM

Diálogo aberto

Estimado aluno,

A relevância dos processadores ARM dentro dos sistemas embarcados mostra como essa arquitetura adequou-se às necessidades do mercado, permitindo a entrega de dispositivos poderosos, com um baixo consumo de energia e alta confiabilidade. Isso fez com que praticamente todos os smartphones da atualidade possuíssem um microprocessador desse tipo. Assim, fica claro que desenvolver projetos utilizando um ARM se tornou um padrão para aplicações que exigem um processamento intermediário ou alto.

Tais características mostram que o planejamento para a realização de um projeto com um processador ARM é um passo fundamental para o desenvolvimento de um sistema desse tipo. Dessa forma, um profissional que trabalha com a implementação de sistemas embarcados deve conhecer ferramentas que permitam o desenvolvimento de um produto eletrônico baseado em um ARM. Somado a isso, também se torna relevante o conhecimento de diferentes aplicações que uma máquina ARM permite, para que, assim, se conheçam as funções e limitações existentes de um dispositivo de tal tipo.

Quando se trata da programação de um microcontrolador, deve-se então conhecer como configurar e utilizar os principais periféricos existentes. Isso tornará possível o interfaceamento com elementos externos, o que tornará a realização de operações de controle, o acionamento de interfaces homem-máquina, o processamento de dados e outras operações, factíveis.

Dentro do contexto que está sendo desenvolvido, você é um desenvolvedor de sistemas embarcados que sempre está apto a trabalhar em novos projetos e desafios. Agora você foi direcionado a trabalhar em um equipamento para a medição da temperatura de uma câmara fria de um frigorífico. O cliente solicitou que o sistema mostrasse a temperatura em um display e ficou a seu encargo integrar esse display ao projeto.

Para esse projeto será utilizado um display LCD alfanumérico em conjunto com um microcontrolador ARM. Cabe a você desenvolver um código que possibilite o interfaceamento do microcontrolador com um display e que permita nele a exibição da temperatura medida.

Como programar o microcontrolador para essa interface? Como é feita a comunicação do microcontrolador com o display? Como montar a lógica para enviar os dados a serem escritos (visualizados) no display?

Para que seja capaz de responder a essas perguntas, você será levado a estudar diferentes aplicações dos processadores ARM, como desenvolver um projeto utilizando um microcontrolador com essa característica e como utilizar simuladores nesse processo. Tudo isso vai torná-lo capaz de desenvolver sistemas variados utilizando um ARM.

Vamos aos estudos?

Não pode faltar

Um sistema existente nos microcontroladores, que é muito utilizado durante a programação, é o de **interrupções**. Esse sistema permite que seja executada uma função indicada pelo programador quando um determinado fato ocorrer no microcontrolador. Praticamente todos os periféricos apresentam estrutura de interrupção do programa. Por exemplo, é possível gerar uma interrupção no estouro da contagem de um *timer* com a recepção de um dado via SPI ou ao final de uma conversão analógico-digital.

Vale ressaltar que uma interrupção gera uma pausa na execução do programa principal para que ela possa ser atendida. Nesse momento, o endereço de memória da próxima instrução do programa principal é armazenado internamente no microcontrolador e este passa a executar as instruções que estão dentro da função de interrupção. Com o fim da execução das instruções da interrupção, o microcontrolador volta a executar as instruções do programa principal a partir do endereço que havia sido armazenado. Assim, fica claro que não devem ser executadas muitas instruções dentro de uma função de interrupção, pois só assim haverá tempo suficiente de processamento dedicado ao programa principal. Tomando-se esse cuidado, é possível evitar que ocorra uma “sobreposição” de interrupções, fazendo com que o microcontrolador se ocupe indefinidamente com a execução de uma longa sequência delas, ou ainda que uma nova condição de interrupção seja gerada no momento em que uma outra ainda não tenha sido finalizada.

Portanto, torna-se evidente que as interrupções são importantes, mas devem ser utilizadas com cautela.

A fim de que essa funcionalidade do microcontrolador seja ativada, é necessário habilitar as interrupções de forma global e realizar a configuração desejada nos registradores do periférico. Uma outra configuração disponível em alguns dispositivos é a prioridade das interrupções, que se torna importante

quando mais de uma está habilitada. Se, por exemplo, for feita a configuração da interrupção A com prioridade maior em relação a B, e se a interrupção A acontecer enquanto a B está sendo executada, a primeira será atendida imediatamente e depois o programa retornará à execução da interrupção B. Caso esta aconteça enquanto a interrupção A está sendo atendida, o microcontrolador irá esperar o fim da execução de A para só então atender a interrupção B.

O periférico mais comum de ter uma interrupção configurada é o *timer*. Dentro de um microcontrolador ARM7 da família STR71x existem quatro *timers*, cada um deles possui um contador principal e dois canais (A e B), que podem ser utilizados para diversas funções. O contador principal possui uma interrupção associada ao “estouro” da contagem de seu registrador de 16 bits, permitindo a implementação de uma função do tipo *delay*, por exemplo. Os canais do *timer* são capazes de realizar capturas ou comparações independentes (e simultaneamente) ou de ser agrupados para gerar um sinal PWM ou um pulso de largura programável (STMICROELETRONICS, 2013).

Com os canais do *timer* operando na função de **captura**, dois dos registradores que compõem esse módulo (uma para cada canal) têm a função de armazenar a contagem do contador principal a cada transição da entrada de captura. A transição pode ser configurada como uma borda de subida ou de descida e está atrelada a um pino do microcontrolador. Nesse modo, também é possível gerar uma interrupção a cada captura, que opera de forma independente para cada canal. Assim, após duas capturas, é possível determinar o período do sinal aplicado no pino pela diferença entre os valores capturados. Para isso, basta calcular o tempo relativo a cada contagem do contador, considerando a sua fonte de *clock* e divisões que foram realizadas conforme a configuração feita.

No modo de **comparação**, a contagem que está sendo realizada pelo contador principal é continuamente comparada com o valor armazenado nos registradores de comparação do canal A e B. Caso ocorra uma coincidência de valores, uma *flag* é ativada, a qual pode gerar uma interrupção e, caso a saída de comparação esteja habilitada, pode-se gerar a mudança no estado de um pino configurado previamente como saída. Com isso, os canais de comparação podem ser utilizados na geração periódica de interrupções e/ou na geração de sinais periódicos. No caso das interrupções, ela se torna conveniente quando é desejado um intervalo entre interrupções diferente do intervalo gerado pelo estouro do contador principal.

Assimile

Dentro de um microcontrolador, uma *flag* tem a função de um sinalizador. Elas são um bit de memória localizados dentro do dispositivo e têm a função de identificar algum evento que aconteceu. Quando

acontece uma interrupção dentro do microcontrolador, são ativadas ao menos duas *flags*, uma dentro do periférico gerador e outra global, indicando a sua ocorrência e causando a pausa na execução do programa principal. Sempre que as *flags* são ativadas, elas devem ser limpas antes ao final da função chamada com a interrupção para que novas interrupções possam ser geradas.

Um terceiro modo de operação dos *timers* é o **OPM** (Modo de Pulso Único, do inglês *One Pulse Mode*), no qual o periférico é utilizado para gerar um pulso de largura programável, disparado por um evento externo. Isso é feito por meio dos módulos de captura e comparação do canal A do *timer*, permitindo que seja gerada uma interrupção no início do pulso, com a detecção da borda de acionamento e, no final, com o fim da contagem do tempo do pulso. Nesse modelo, ao ser detectada uma borda de acionamento no pino correspondente, o conteúdo do contador é reiniciado e o pino de saída, atrelado ao módulo, muda de estado. A contagem segue até que se atinja um valor programado, quando o pino volta ao estado inicial, gerando um pulso com a largura programada pela contagem. Como esse módulo utiliza somente o canal A do timer, o canal B fica disponível para funcionar em outro modo.

O modo de operação **PWM** é aplicado no controle de dispositivos externos, no qual o *timer* é configurado para gerar um sinal periódico, com frequência fixa e ciclo ativo variável, entre 0 % e 100 %. Para o sinal ser gerado a contagem do *timer* é continuamente comparada com o conteúdo de dois registradores. Quando ela atinge o valor do primeiro registrador, o pino de saída associado ao canal A muda de estado. Ao atingir o valor do segundo registrador, o pino volta ao seu estado inicial e a contagem é reiniciada. Não existem interrupções associadas diretamente a esse modo de operação.

Exemplificando

Para compreender como é realizada a configuração e a utilização de uma interrupção, vamos analisar o código a seguir, que demonstra a utilização da interrupção de estouro da contagem do timer0 para o controle de um pisca de LED.

```
1 //***** Função que trata a interrupção do timer *****
2 __irq __arm void trata_timer0_tof(void)
3 {
4     GPIOPD ^= BIT12 + BIT13; // Inverte o estado dos pinos P0.12 e P0.13
5     TIM0_SR_bit.TOF = 0;      // Apaga a flag de interrupção do TIMERO
6     EIC_IPR = bIRQ29;        // Apaga a flag de interrupção do
```

```

7 }                                     // controlador de interrupções
8
9 //***** Função de inicialização dos periféricos *****
10 void inicializa (void)
11 {
12     // Coloca os pinos P0.12 e P0.13 como saídas push-pull
13     CONFIG_PORT_OUT(0,BIT12+BIT13);
14     GPIO0PD = BIT12;           // Liga o LED do pino P0.12
15     // Configura o controlador de interrupções com o endereço da função
16     // que trata da interrupção
17     EIC_IVR = (long)trata_timer0_tof;
18     EIC_SIR29 = ((long)trata_timer0_tof<<16);
19     EIC_SIR29 |= 1;           // Define a interrupção do timer0 com prioridade 1
20     EIC_IER = bIRQ29;         // Habilita a interrupção de estouro do timer0
21     EIC_ICR = bIRQ_EN;        // Habilita as interrupções no microcontrolador
22     TIM0_CR1 = bEN;          // Habilita o timer0
23     TIM0_SR = 0;             // Limpa todas as flags do timer0
24     TIM0_CR2 = bTOIE + 29;    // Habilita a interrupção de estouro da
25 }                               // contagem e prescaler para dividir o clock por 30 (20+1)
26
27 //***** Função principal *****
28 void main (void)
29 {
30     inicializa();            // Chama a função de inicialização
31     while (1) {}             // Simplesmente aguarda a interrupção
32 }

```

Fonte: adaptado de Pereira (2007, p. 238).



Para melhor visualizar esse código, acesse o QR Code ou o link:
https://cm-cls-content.s3.amazonaws.com/ebook/embed/qr-code/2020-2/sistemas_digitais_microprocessadores/u4/lklis202_u4s3_sis_dig_QRCODE3.pdf

O grupo de registradores EIC é responsável por configurar as interrupções de forma global no microcontrolador e deve ser configurado para permitir o funcionamento de qualquer interrupção. No caso da interrupção de estouro do timer0, ela é ordenada como sendo a interrupção de número 29. Por isso, aparecem registradores com esse número (PEREIRA, 2007). Perceba que, para o funcionamento da interrupção, ela é configurada em três locais. Primeiramente, é habilitada a interrupção de estouro do timer0 no registrador de controle de interrupções. Depois, são habili-

tadas as interrupções de forma global no microcontrolador. E, por fim, ela é habilitada no registrador do próprio periférico.

Repare ainda que a função principal (*main*) executa a inicialização dos periféricos e depois entra em um *loop* infinito (*while(1)*). Assim, o programa ficará preso nesse loop, até o momento em que houver o estouro da contagem do *timer0*, que deve acontecer a cada 1,02 Hz conforme as configurações de *prescaler* realizadas. Quando a interrupção ocorrer, acontece a execução da função indicada. Dentro dessa função, a operação desejada é realizada e no seu final as flags de interrupção do estouro do *timer0* são limpas, permitindo que uma nova interrupção venha a acontecer.

Um outro periférico bastante utilizado em microcontroladores é o **conversor analógico-digital** (A/D). O microcontrolador ARM da família STR71x possui um conversor de doze bits com quatro canais de entrada e pode operar de dois modos, sequencial e em canal único. No modo sequencial, cada canal é amostrado e convertido sequencialmente em uma operação que demanda um total de 2048 ciclos de *clock* do modulador. No modo de canal único, um único canal é amostrado e convertido em 512 ciclos de *clock*, porém o dado do canal é atualizado a cada 2048 ciclos, mesmo tempo que ocorre no modo sequencial (PEREIRA, 2007).

Nesse periférico, quando a conversão for concluída, uma *flag* é ativada, indicando que um novo dado se encontra disponível para leitura em seu registrador específico. Assim, é possível configurar o conversor para gerar uma interrupção sempre que uma conversão foi finalizada em um determinado canal.

Também é possível associar interrupções aos pinos de entradas e saídas do microcontrolador, com o chamado módulo de interrupções **externas**. Nos microcontroladores STR71x estão disponíveis catorze fontes de interrupções associadas aos pinos, que permitem a identificação de um evento de borda de subida ou descida. Assim, quando o evento é detectado, a função associada à interrupção é chamada.

Se comparadas às demais interrupções, as externas são as mais fáceis de serem configuradas, bastando configurar o pino como entrada, selecionar o tipo de evento, habilitar a interrupção associada a ele e habilitar as interrupções do microcontrolador. Essa simplicidade faz com que esse módulo seja muito utilizado para detectar quando botões são pressionados, evitando a verificação constante da mudança de estado do pino no código principal.

Além dos periféricos citados, é possível configurar interrupções nos **módulos de comunicação**, sejam eles o SPI, I²C, USART ou USB,

independentemente do microcontrolador. Os eventos mais comuns, que podem ser encontrados para esses periféricos, são aqueles que indicam se os *buffers* de transmissão ou recepção estão cheios ou vazios. Dependendo do microcontrolador e do protocolo, ainda podem ser encontradas interrupções que indicam erros de transmissão ou temporização dos dados.

Sabendo como funcionam as interrupções, você já tem uma visão completa de como é realizada a programação de um microcontrolador, conhecendo os periféricos existentes e configurações possíveis. Assim, é importante saber como planejar o projeto de um sistema microcontrolado.

Depois de selecionado qual microcontrolador utilizar em um projeto, você deve determinar quais elementos, além dele, serão utilizados. Se você está realizando a comunicação com um sensor SPI, já tenha em mente que será necessário configurar esse periférico. Se existe alguma interface homem-máquina, você deve reservar os pinos para alocar botões e LEDs sinalizadores, por exemplo. Caso haja a necessidade de realizar a leitura de algum sinal analógico, você precisará do conversor A/D. Dessa forma, é possível determinar todos os periféricos a serem utilizados.

Determinados os periféricos, chega a parte de configurá-los e, para isso, você pode utilizar bibliotecas, evitando o acesso direto aos registradores. Dependendo da aplicação, será necessário utilizar interrupções, as quais poderão tornar o seu projeto mais otimizado. Para determinar isso, faça a programação por partes. Se, por exemplo, você deve realizar a leitura de um sinal analógico e armazenar os valores lidos em memória externa, primeiro configure e coloque em funcionamento o conversor A/D. Só depois adicione a comunicação com a memória externa no seu *firmware*.

Outro ponto importante em um projeto que controla diversas operações sequenciais é a forma como ele é estruturado. A melhor opção nesse caso é pensar no programa como uma grande máquina de estados baseada em uma lógica de *switch/case*. Assim, algumas ações são realizadas em um estado e só se passa para o próximo se determinada condição for satisfeita.

É importante saber que o tipo de variável que está sendo utilizada impacta na velocidade de processamento. Se você está realizando operações de valores sem sinal, sempre utilize a diretiva “*unsigned*” para declarar as variáveis. Além disso, considere o tamanho das variáveis conforme a largura da memória do microcontrolador que está sendo utilizado. Um ARM de 32 bits realiza a execução de códigos com variáveis de 32 bits mais rápido do que com variáveis de 8 bits (PEREIRA, 2007).

Como já foi tratado em outras seções, para se realizar a programação de um microcontrolador, é necessário utilizar uma interface de programação (IDE) e,

para uma situação em que não se tenha o dispositivo físico em mãos, a possibilidade de simulação se torna uma ferramenta interessante. Para os microcontroladores ARM, uma opção de IDE que traz essa funcionalidade é o Keil MDK. Assim, é possível testar o passo a passo da execução de um programa, avaliando os valores contidos em registradores e armazenados nas variáveis. Com isso, tem-se uma previsão da operação do programa desenvolvido.

Uma outra opção de simulador que pode ser utilizado é o Proteus. Esse software realiza a simulação de circuitos elétricos e eletrônicos, permitindo que seja adicionado um microcontrolador e que seja vinculado um programa previamente compilado nele. Assim, é possível simular a interação do microcontrolador com elementos externos como LEDs, sensores e outros dispositivos, utilizando ferramentas para a análise de sinais como osciloscópios virtuais e telas de *display*.

Refita

Você acredita que a utilização de um simulador ajuda no desenvolvimento de um *firmware* para microcontrolador? Mesmo tendo o dispositivo fisicamente disponível, ainda vale a pena utilizar o simulador?

Conhecendo a forma que os periféricos de um microcontrolador são configurados e utilizados, você já tem conhecimento suficiente para programar um microcontrolador de forma eficiente. Assim, você pode começar criando programas em uma IDE que permita a simulação do código desenvolvido, para assim praticar e se tornar um profissional capacitado para o desenvolvimento de sistemas embarcados. Dessa forma, finalizamos esta seção, tornando você capaz de aplicar a programação para o desenvolvimento de sistemas com microprocessadores e microcontroladores.

Sem medo de errar

Dentro do contexto em que você é um desenvolvedor de sistemas embarcados, cabe a você desenvolver um código para permitir a interface de um microcontrolador ARM com um *display*. Isso faz parte de um equipamento para a medição da temperatura de uma câmara fria e irá permitir que a temperatura seja verificada visualmente.

Para o desenvolvimento desse projeto, será utilizado o microcontrolador STR711, fabricado pela STMicroelectronics, que possui uma CPU ARM7 de 32 bits em conjunto com um *display* LCD de dezesseis caracteres por duas linhas. Devido à maior disponibilidade no mercado nacional, optou-se por utilizar um *display* alimentado com 5 V, ao passo que o microcontrolador é

alimentado com 3,3 V. Dessa forma, para evitar a necessidade de circuitos externos, escolheu-se realizar somente operações de escrita no display. Isso é possível pois o controlador presente no módulo display, o HD44780, fabricado pela Hitachi, possui entradas compatíveis com níveis TTL, permitindo que um nível de 3,3 V seja entendido como alto sem problemas.

Nesse projeto, a interface do microcontrolador com o display foi feita por seis pinos, todos configurados como saídas digitais do microcontrolador, fazendo parte do bloco de pinos GPIO0. Destes pinos, quatro são utilizados para a transferência de dados (pinos de 4 a 7), um para habilitar o display (pino 0), ativo em nível alto, e um para indicar se estarão sendo transmitidos dados (nível alto) ou comandos (nível baixo) ligados ao pino 1. Caso fossem realizadas operações de leitura e escrita no display, seria necessário mais um pino. Como isso não aconteceu nesse projeto, o pino do display relativo a essa informação foi ligado diretamente ao terra, indicando uma operação de escrita.

Como o objetivo é implementar somente a comunicação com o display, é possível elaborar uma biblioteca para tal fim, com diversas funções para o envio das informações. No funcionamento do display, é possível selecionar a posição em que os dados serão escritos por meio de um comando específico. No caso dos dados, ao serem enviados, eles já serão escritos no display, conforme uma tabela ASCII (Código Padrão Americano para o Intercâmbio de Informação, do inglês *American Standard Code for Information Interchange*).

Sabendo disso, o código a seguir mostra algumas funções que podem ser criadas:

```
1 //***** Função de envio de um nibble para o display *****
2 /* dado: dado a ser enviado para o display (somente o nibble menos significativo)
3 //*****
4 void lcd_envia_nibble(char dado)
5 {
6     lcd_port &= 0xFF0F;           // Coloca nível baixo nos pinos de dados
7     lcd_port |= dado << 4;       // Coloca os dados nos pinos
8     GPIO0PD |= BIT0;            // Habilita o pino de enable
9     GPIO0PD &= ~BIT0;           // Desabilita o pino de enable
10 }
11
12 //***** Função de escrita de 1 byte no display*****
13 /* endereco : 0 se instrução, 1 se dado
14 /* dado : dado ou comando a ser escrito
15 //*****
16 void lcd_envia_byte(char endereco, char dado)
17 {
```

```

18     if (endereco)           // Configura se será enviada uma instrução ou dado
19         GPIO0PD |= BIT1;    // Coloca o pino de RS em nível alto
20     else
21
22         GPIO0PD &= ~BIT1; // Coloca o pino de RS em nível baixo
23
24         delay_ms(1);       // Temporização necessária para o processamento do display
25         GPIO0PD &= ~BIT0;      // Desativa o enable do display
26         lcd_envia_nibble(dado >> 4); // Envia os bits mais significativos do dado
27         lcd_envia_nibble(dado & 0x0F); // Envia os bits menos significativos do dado
28     }
29
30 //*****Função de posicionamento do cursor do display *****
31 /* x : coluna a ser posicionado o cursor (iniciando de 1)
32  * y : linha a ser posicionado o cursor (1 ou 2)
33 //*****
34 void lcd_pos_xy(char x, char y)
35 {
36     char endereco;           // Define a variável para salvar a posição
37     if (y!=1)                // Verifica a linha
38         endereco = 0x40;      // Configura o endereço para a segunda linha
39     else
40         endereco = 0;        // Configura o endereço para a primeira linha
41     endereco += x-1;         // Ajusta a posição conforme a coluna
42     lcd_envia_byte(0, 0x80 | endereco); //Envia o endereço
43 }

```

Fonte: adaptado de Pereira (2007, p. 407).



Para melhor visualizar esse código, acesse o QR Code ou o link: https://cm-cls-content.s3.amazonaws.com/ebook/embed/qr-code/2020-2/sistemas_digitais_microprocessadores/u4/lkl202_u4s3_sis_dig_mic_QRCODE4.pdf

Repare que, para a configuração da posição, o display considera apenas um único byte, onde os endereços de 0x80 a 0x8F são as posições da primeira linha e os endereços de 0xC0 a 0xCF são as posições da segunda, onde a contagem das posições vão da esquerda para a direita. Além disso, é importante saber que o controlador do *display* faz o incremento automático dos endereços. Assim, para se escrever uma *string* de dados basta enviá-los sequencialmente com atenção para que não estourem a quantidade de dígitos representados no display. Repare ainda que para o funcionamento da biblioteca é necessária uma função *delay*, implementada com base de tempo de milissegundos.

Utilizando as funções apresentadas em conjunto com a inicialização dos pinos utilizados na comunicação, é possível criar a biblioteca para ser utilizada no projeto. Com isso, ela poderia ser integrada no programa principal do sistema de medição da temperatura.

Avançando na prática

Configuração de conversor A/D de um ARM

Você trabalha como desenvolvedor de sistemas embarcados e foi incumbido de desenvolver o *firmware* de um microcontrolador ARM para um sistema de monitoramento do sinal de um sensor analógico de pressão. O sistema em questão deve acender um LED quando a tensão do sensor estiver próxima a 1,25 V, indicando que o processo monitorado está funcionando na pressão adequada. Como desenvolver um código para obter a funcionalidade desejada?

Resolução da situação-problema

Como o sensor de pressão é analógico, será necessário configurar o conversor A/D do microcontrolador, no qual os valores lidos serão comparados a uma faixa de tolerância para verificar se o LED deve ser aceso ou não. O microcontrolador utilizado será o STR711, que possui um conversor A/D bipolar de 12 bits. Esse módulo conversor possui ponto central de conversão localizado em uma referência de tensão interna, tipicamente no valor de 1,25 V \pm 5%. Portanto, tensões analógicas de entrada inferiores a isso irão gerar um número binário negativo e superiores a um número positivo, representados em complemento de dois. Com isso, uma faixa aceitável para o acendimento do LED é quando o resultado da conversão estiver entre -16 (0xFF0 e 12 (0x00F). Dessa forma, um código pode ser elaborado para implementar o sistema como apresentado a seguir:

```
1 unsigned int result;           // Declara a variável para armazenar o dado da
2                                         // conversão
3 void main (void)
4 {
5     CONFIG_PORT_OUT(0, BIT13);   // Pino P0.13 como saída push-pull
6     CONFIG_PORT_AIN(1,BIT0);    // Configura o pino P1.0 para a função analógica
7     PCU_BOOTCR = bADCEN;       // Habilita o conversor A/D
8     ADC_CPR = 2;               // Prescaler do ADC = PCLK2 / 4 = 500 kHz
9     ADC_CSR = bSINGLE_CHAN + bADC_CH0; // modo de canal único, canal 0
```

```

10    while (1)
11    {
12        if (ADC_CSR & bADC_DA0) // Se o bit DAO do conversor estiver setado
13        {                      //      uma nova conversão está disponível 4
14            result = ADC_DATA0 >> 4; // Lê o resultado e desloca 4 bits
15            if ((result>=0xFF0 && result <=0xFFFF) || (result<=0x00F)) // Está na faixa?
16                GPIO0PD=BIT13; // Se sim, liga o LED
17            else
18                GPIO0PD = 0; // Se não, apaga o LED
19            ADC_CSR_bit.DAO = 0; // Apaga o flag DAO
20        }
21    }
22 }

```

Fonte: adaptado de Pereira (2007, p. 291).



Para melhor visualizar esse código, acesse o QR Code ou o link: https://cm-cls-content.s3.amazonaws.com/ebook/embed/qr-code/2020-2/sistemas_digitais_microprocessadores/u4/lklis202_u4s3_sis_dig_mic_QRCODE5.pdf

No código, perceba que, para configurar os pinos, foi utilizada uma biblioteca, evitando o acesso direto aos registradores. Além disso, é possível identificar que a comparação e a leitura do valor da conversão são feitas somente quando o bit DA0 vai a 1, o que indica que a conversão foi finalizada e os dados estão disponíveis para leitura. Assim, com o código apresentado, a funcionalidade desejada para o projeto é obtida.

Faça valer a pena

1. Dentro do funcionamento de um microcontrolador, as interrupções possuem um papel relevante, permitindo que algoritmos complexos sejam desenvolvidos, o que traz um processamento diferenciado especialmente em sistemas de controle.

Sobre as interrupções, assinale a alternativa correta.

- Elas têm a função de estabelecer contagens de tempo precisas, que podem ser configuradas conforme a necessidade do programador.
- Elas são capazes de realizar a conversão de um sinal analógico para digital, sendo multiplexada em vários canais dentro do microcontrolador.

- c. Elas permitem que seja executada uma função indicada pelo programador quando um determinado evento acontecer no microcontrolador.
- d. Elas são capazes de estabelecer a comunicação serial com um ou mais dispositivos externos.
- e. Elas permitem o envio de dados criptografados via USB para qualquer dispositivo conectado.

2. Um periférico bastante comum de ser encontrado em um microcontrolador é o *timer*. Em um ARM podem ser encontrados vários *timers* e cada um deles possui um contador principal e dois canais (A e B) que podem ser utilizados para diversas funções. Sobre os *timers*, considere as seguintes afirmações:

- I. Com os canais do *timer* operando na função de captura, dois dos registradores que compõem esse módulo (uma para cada canal) têm a função de armazenar a contagem do contador principal a cada transição da entrada de captura.
- II. No modo de comparação, a contagem que está sendo realizada pelo contador principal é continuamente comparada com o valor armazenado nos registradores de comparação do canal A e B.
- III. No modo de operação, o OPM, em que o periférico é utilizado para gerar um pulso de largura programável, é disparado por um evento externo.
- IV. O modo de operação **PWM** é aplicado no controle de dispositivos externos, onde o *timer* é configurado para gerar um sinal periódico, com frequência fixa e ciclo ativo variável, entre 0% e 100%.

Assinale a alternativa que indica quais afirmativas estão corretas.

- a. Somente I e III.
- b. Somente III e IV.
- c. Somente I, II e IV.
- d. Somente II, III e IV.
- e. I, II, III e IV.

3. É possível configurar diversas interrupções em um microcontrolador para os mais variados periféricos. Sobre o assunto, considere as seguintes afirmações:

- () É possível associar interrupções aos pinos de entradas e saídas do microcontrolador.

() Algumas interrupções que podem ser configuradas em periféricos de comunicação são para indicar se os *buffers* de transmissão ou recepção estão cheios ou vazios.

() É possível configurar o conversor A/D para gerar uma interrupção sempre que uma conversão for finalizada em um determinado canal.

() Para que as interrupções sejam habilitadas, o microcontrolador deve receber um byte de configuração via I²C.

Classifique as afirmativas em verdadeiro (V) ou falso (F) e assinale a alternativa que indica a classificação correta, respectivamente.

- a. F – V – V – F.
- b. F – V – F – V.
- c. V – F – V – F.
- d. V – F – F – V.
- e. V – V – V – F.

Referências

- ARDUINO. **digitalWrite ()**. [s.d.]. Disponível em: <https://www.arduino.cc/en/Reference/digitalWrite>. Acesso em: 6 out. 2020.
- FORTE, C. H. V. **Processadores ARM: visão geral e aplicações**. Departamento de Ciência da Computação e Estatística – Universidade Estadual Paulista Júlio de Mesquita Filho (UNESP), São José do Rio Preto, 2015.
- GARCIA, F. D. Breve histórico da ARM. **Embarcados**, [S.I.], 22 mar. 2017. Disponível em: <https://www.embarcados.com.br/breve-historico-da-arm>. Acesso em: 3 jun. 2020.
- LUGER, G. F. **Inteligência Artificial**. 6. ed. São Paulo: Pearson Education do Brasil, 2013.
- MEDEIROS, L. F. de. **Inteligência Artificial aplicada: uma abordagem introdutória**. Curitiba: InterSaber, 2018.
- MICROCHIP. **ATmega48A/PA/88A/PA/168A/PA/328/P**. [S.l.: s.n.], 2018. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf>. Acesso em 20 mai. 2020.
- MICRON. **M25P10A 1Mb 3V Serial Flash Embedded Memory** – datasheet. [S.l.: s.n.], 2014. Disponível em: <https://www.micron.com/-/media/client/global/documents/products/data-sheet/nor-flash/serial-nor/m25p/m25p10a.pdf>. Acesso em: 27 maio 2020.
- MIYADAIRA, A. N. **Microcontroladores PIC18**: aprenda e programa em linguagem C. 4. ed. São Paulo: Érica, 2013.
- MONK, S. **30 projetos com Arduino**. 2. ed. Porto Alegre: Bookman, 2014.
- MONK, S. **Programação com Arduino**: começando com sketches. 2. ed. Porto Alegre: Bookman, 2017. *E-book*.
- OLIVEIRA, C. V.; ZANETTI, H. P. **Arduino Descomplicado** – Como Elaborar Projetos de Eletrônica. São Paulo: Érica, 2015.
- PEREIRA, F. **Microcontroladores PIC** – técnicas avançadas. São Paulo: Érica, 2002.
- PEREIRA, F. **Tecnologia ARM**: microcontroladores de 32 bits. São Paulo: Érica, 2007.
- STMICROELECTRONICS. **STR71xFxx STR710RZ datasheet**. [S.l.: s.n.], 2013. Disponível em: <https://www.st.com/resource/en/datasheet/str711fr0.pdf>. Acesso em: 27 maio 2020.

