ERCEMAPI

Escola Regional de Computação
do Ceará, Maranhão e Piauí

Livro de Minicursos

14 a 16
Setembro 2021
~~Quixadá, CE, Brasil~~
Evento Virtual

Realização: SBC

Organização: Universidade Federal do Ceará; Instituto Federal Ceará; UESPI; Universidade Federal do Piauí; UFMA

Apoio: PET Sistemas de Informação; PET Tecnologia da Informação

Patrocínio: Loggi

IX Escola Regional de Computação Ceará, Maranhão e Piauí
14 a 16 de setembro de 2021

LIVRO DE MINICURSOS
ERCEMAPI 2021

ORGANIZAÇÃO DO LIVRO:

CARLA ILANE MOREIRA BEZERRA (UFC - QUIXADÁ)

DAVI VIANA (UFMA)

ALCEMIR RODRIGUES SANTOS (UESPI)

COORDENAÇÃO GERAL:

DANIELO GONÇALVES GOMES (UFC)

MARIA VIVIANE DE MENEZES (UFC – QUIXADÁ)

Sociedade Brasileira de Computação – SBC

Porto Alegre

2021

Capa: Diagramação UFC Quixadá
Editoração: Carla Ilane Moreira Bezerra e Alcemir Rodrigues Santos

Dados Internacionais de Catalogação na Publicação (CIP)

E74 Escola Regional de Computação do Ceará, Maranhão e Piauí (9. : 2021 : Quixadá, CE)
Minicursos da 9ª Escola Regional de Redes de Computador [recurso eletrônico] – Organizadores: Carla Ilane Moreira Bezerra, Davi Viana, Alcemir Rodrigues Santos – Porto Alegre : SBC, 2022.

ISBN 978-65-87003-73-3

1. Computação. 2. Inteligência artificial. 3. Propriedade intelectual. 4. Arquitetura de computadores I. Gomes, Danielo Gonçalves. II. Menezes, Maria Viviane de. III. Título.

CDU 004

Catalogação elaborada por Francine Conde Cabral
CRB-10/2606

SINOPSE EM PORTUGUÊS

O Livro de Minicursos da ERCEMAPI 2021 colabora com o objetivo da Escola Regional de Computação Ceará, Maranhão e Piauí em disseminar o conhecimento técnico e científico sobre temas e assuntos de vanguarda na área de Computação. Em sua 9ª edição, os minicursos abordam conteúdos relacionados à inteligência artificial, propriedade intelectual e arquitetura de computadores, como forma de atualizar os conhecimentos da comunidade acadêmica e profissional, de uma forma didática e de amplo acesso ao público. Os três capítulos deste livro possuem metodologias e ferramentas para a área de Tecnologia da Informação e Comunicação, sendo uma excelente oportunidade para a familiarização dos interessados com novos temas de pesquisa que podem vir a ser úteis em suas vidas profissionais.

SINOPSE EM INGLÊS

ERCEMAPI 2021 Short Course Book collaborates with the goal of the Ceará, Maranhão and Piauí Regional Computing School in disseminating technical and scientific knowledge on cutting-edge topics and subjects in the field of Computing. Now in its 9th edition, the short courses address content related to artificial intelligence, intellectual property and computer architecture, to update the knowledge of the academic and professional community in a didactic way with broad access to the public. Three chapters of this book have methodologies and tools for the Information and Communication Technology area, an excellent opportunity to familiarize those interested with new research topics that may be useful in their professional lives.

Mensagem da Coordenação de Minicursos

O Livro de Minicursos da ERCEMAPI 2021 colabora com o objetivo da Escola Regional de Computação Ceará, Maranhão e Piauí em disseminar o conhecimento técnico e científico sobre temas e assuntos de vanguarda na área de Computação. Em sua 9ª edição, os minicursos abordam conteúdos relacionados ao processamento de linguagem natural, propriedade intelectual e à arquitetura e organização de computadores, como forma de atualizar os conhecimentos da comunidade acadêmica e profissional, de uma forma didática e de amplo acesso ao público. Os três minicursos foram selecionados através de um processo de revisão por pares do tipo *"blind"*, de um total de 10 propostas de minicursos submetidas, o que implicou uma taxa de aceitação de 30%.

No **Capítulo 1**, "PLN: Das técnicas tradicionais aos modelos de *deep learning*", os autores perpassam os conceitos do processamento de linguagem natural fazendo uma revisão desde as primeiras técnicas desenvolvidas até as técnicas mais modernas.

O **Capítulo 2**, "Propriedade Intelectual e Registro de Programa de Computador – Inovação e Tecnologia na Indústria Moderna", os autores apresentam um histórico do desenvolvimento da propriedade intelectual no Brasil, apresentando o processo de registro de marcas, patentes, bem como de programas de computadores.

Já o **Capítulo 3**, "Uma Abordagem Prática Para Aprendizagem em Arquitetura e Organização de Computadores com Apoio do Simulador Computacional CompSim", os autores justificam a necessidade de simulação para apoiar o aprendizado prático em projetos de sistemas computacionais e apresentam o ambiente de simulação do projeto de novos sistemas computacionais virtuais, o CompSim.

Os três capítulos deste livro possuem técnicas, processos e ferramentas úteis para o desenvolvimento da carreira profissional dos leitores interessados nestes tópicos da ciência da computação.

Agradecemos a todos os professores que compuseram o Comitê de Programa de Minicursos, por suas valiosas contribuições para os trabalhos e dedicação no

processo de revisão, em especial aos coordenadores gerais da ERCEMAPI 2021, Prof. Danielo Gomes (UFC) e Profa. Maria Viviane de Menezes (UFC – Quixadá), por todo o suporte durante a seleção e elaboração deste livro. Finalmente, agradecemos a todos os participantes dos minicursos da ERCEMAPI 2021 realizados mais uma vez de forma remota.

Carla Ilane Moreira Bezerra (UFC – Quixadá)
Davi Viana (UFMA)
Alcemir Rodrigues Santos (UESPI)

Coordenadores de Minicursos da ERCEMAPI 2021

Comitê de Programa de Minicursos

Áurea Hiléia da Silva Melo (UEA)
Dario Brito Calçada (UESPI)
Emanuel Ferreira Coutinho (UFC)
Fábio Carlos Sousa Dias (UFC)
Francisco Airton Pereira da Silva (UFPI)
Ismayle de Sousa Santos (UFC)
Jeandro de Mesquita Bezerra (UFC)
José Maria da Silva Monteiro Filho (UFC)
Maicon Bernardino da Silveira (Unipampa)
Marcela Sávia Picanço Pessoa (UEA)
Marcia Henke (UFSM)
Michel Sales Bonfim (UFC)
Wladimir Araújo Tavares (UFC)

Secretarias Regionais SBC

Danielo Gonçalves Gomes (UFC / SR Ceará)
Davi Viana (UFMA / SR Maranhão)
Rodrigo Augusto Rocha Souza Baluz (UESPI / SR Piauí)

Sumário

Capítulo 1

PLN: Das Técnicas Tradicionais aos Modelos de Deep Learning

Rafael Anchiêta, Francisco A. R. Neto, Jeziel C. Marinho, Raimundo Moura

Abstract

With the massive amount of data generated daily on the Web, researchers in the field of Natural Language Processing have focused on extracting useful information from unstructured data. This volume of data makes it impractical for anyone to manually process them in order to extract meaningful information, i.e., feelings, opinions, irony, hate speech, fake news, and others. The main objective of this short course is to introduce principles, traditional techniques, and tools in the field of NLP, developing models for binary classification tasks. The course focuses on practical activities using the Python language and libraries such as: NLTK, SpaCy, and Scikit-Learn. In the final part, some topics about Deep Learning will be discussed, including the BERT language model.

Resumo

Com a imensa quantidade de dados gerados diariamente na Web, pesquisadores da área de Processamento de Linguagem Natural (PLN) têm buscado extrair informações úteis de dados não estruturados. Esse volume de dados torna impraticável para qualquer pessoa processá-los manualmente a fim de extrair informações significativas, i.e., sentimentos, opiniões, ironia, discurso de ódio, fake news, entre outros. O objetivo principal deste minicurso é apresentar princípios, técnicas tradicionais e ferramentas da área de PLN, desenvolvendo modelos para tarefas de classificação binária. O curso é focado em atividades práticas usando a linguagem Python e bibliotecas, como: NLTK, SpaCy e Scikit-Learn. Na parte final, alguns tópicos sobre Deep Learning serão discutidos, incluindo o modelo de língua BERT.

1.1. Introdução

Processamento de Linguagem Natural (PLN) é uma vertente da Inteligência Artificial (IA) que ajuda computadores a entender, interpretar e manipular a linguagem humana. Em

termos simples, [Sarkar 2019] define linguagem natural como sendo uma linguagem desenvolvida e evoluída por humanos por meio do uso natural e da comunicação, em vez de construir e criar a linguagem artificialmente, como uma linguagem de programação.

A Comissão Especial de Processamento de Linguagem Natural (CE-PLN) da Sociedade Brasileira de Computação (SBC), estabelece que a área de PLN, também denominada Linguística Computacional ou, ainda, Processamento de Línguas Naturais, busca investigar, propor e desenvolver formalismos, modelos, técnicas, métodos e sistemas computacionais para resolver problemas relacionados à automação da interpretação e da geração da língua humana, como o inglês ou o português. A CE-PLN também descreve que as principais aplicações envolvem áreas, tais como: Tradução Automática de Textos, Sumarização Automática, Ferramentas de Auxílio à Escrita, Perguntas e Respostas, Categorização Textual, Recuperação e Extração de Informação, Análise Morfo-sintática e Análise Semântica, e Análise de Sentimentos.

Em um curso realizado em 2012 na Universidade de Stanford[1], Dan Jurafsky e Chris Manning classificaram as tarefas de PLN em:

- **Resolvidas**: detecção de *spams*, *POS tagging*, reconhecimento de entidades nomeadas;
- **Bom progresso**: análise de sentimentos, resolução de correferência, desambiguação lexical de sentidos, análise sintática (*parsing*), tradução automática, extração de informações;
- **Difíceis**: perguntas e respostas, paráfrases, sumarização, diálogos.

Hoje, cerca de 10 anos após o curso, essa classificação ainda é considerada válida, em especial para o português, uma língua considerada de poucos recursos (*low-resource language*). No entanto, o Centro de Pesquisa para Inteligência Artificial Avançada no Brasil[2] possui um desafio denominado NLP2 para produzir recursos e levar o Processamento de Linguagem Natural em Português para o estado-da-arte nas pesquisas mundiais.

Destaca-se que o foco principal deste minicurso é a tarefa de Análise de Sentimentos e Mineração de Opinião, tendo como insumo básico descrições textuais. De modo geral, a tarefa de Análise de Sentimentos (AS) pode ser definida como qualquer estudo feito computacionalmente envolvendo opiniões, sentimentos, avaliações, atitudes, afeições, visões, emoções e subjetividade, expressos de forma textual [Liu 2012, Liu 2015]. A tarefa de AS pode ser estruturada em três etapas: i) identificar as opiniões expressas sobre determinado assunto ou alvo em um conjunto de documentos; ii) classificar a orientação semântica ou a polaridade dessa opinião em positiva ou negativa; e iii) apresentar os resultados de forma agregada e sumarizada.

É importante mencionar que a maioria das pesquisas da área de AS são baseadas no nível de palavra, através da exploração de padrões linguísticos em tuplas, como <Caracteristica; PalavraOpinativa>. No entanto, métodos baseados no nível de conceito

[1]Natural Language Processing. Lecture Slides. Disponível em: https://web.stanford.edu/ jurafsky/NLPCourseraSlides.html

[2]C4AI: http://c4ai.inova.usp.br/pt/home-2/

têm surgido e precisam ser melhor investigados. Além disso, o uso de *Deep Learning* como as Redes Neurais Convolucionais [Poria et al. 2016] tornou-se muito importante para a evolução das pesquisas na área.

Além da classificação de sentimentos, outras tarefas têm, recentemente, ganho ênfase pela comunidade científica, tais como: detecção de *fake news* e identificação de discurso de ódio, por exemplo. A primeira foca em identificar conteúdo enganoso divulgado principalmente na Web, enquanto a segunda enfatiza a identificação e o combate a um tipo de linguagem que não é aceitável e é prejudicial as pessoas. Assim como na tarefa de classificar sentimentos, os primeiros trabalhos que se propuseram a lidar com notícias falsas e discurso de ódio focaram em abordagens baseadas em léxico, ou seja, um vocabulário pré-definido para ajudar na classificação dessas tarefas [Meneses Silva et al. 2021, Fortuna and Nunes 2018]. Posteriormente, os métodos evoluíram para abordagens baseadas em *deep learning* e modelos de língua, alcançando resultados superiores [Leite et al. 2020, Kaliyar et al. 2021].

No âmbito da Universidade Federal do Piauí (UFPI), pesquisadores do Laboratório de Processamento de Linguagem Natural (LPLN) têm desenvolvido estudos com o uso de PLN, a saber: i) identificação de elementos da Linguagem de Modelagem Unificada (UML) a partir de descrições textuais escritas em Português do Brasil [Anchiêta et al. 2013]; ii) metodologia para auxiliar a escrita de documentos de especificação de requisitos de sistemas [Soares and Moura 2015]; iii) estudo comparativo sobre métodos estatísticos de extração características em descrições textuais, usados para classificação de sentimentos [Anchiêta et al. 2015]; iv) definição da abordagem TOP(X) para inferir os comentários mais úteis sobre produtos e serviços [Sousa 2015]; v) variações da abordagens TOP(X), explorando as variáveis de entrada reputação do autor, quantidade de túplas <característica; palavra opinativa> e riqueza do vocabulário, além de variações dos modelos matemáticos utilizados: Sistema Fuzzy e Redes Neurais Artificiais [Santos et al. 2021]; e vi) estudo de técnicas de PLN para a resolução de problemas de regulação médica em planos de saúde [Magalhães Jr et al. 2019], cuja ideia explorou a descrição textual de prescrições de exames médicos para introduzir um fator de confiança e auxiliar a definição das regulações a serem aprovadas automaticamente.

Neste contexto, o objetivo principal deste minicurso é apresentar princípios, técnicas e ferramentas de PLN para a criação de modelos voltados para a classificação textos. O curso apresenta uma discussão de conceitos, com foco em atividades práticas usando a linguagem Python e bibliotecas que auxiliam no processamento de língua natural, como: NLTK (*Natural Language Toolkit*) [Bird 2006], SpaCy[3] e Scikit-Learn [Pedregosa et al. 2011]. Ao longo do curso, os trechos de código discutidos mostram o resultado da execução usando a tag '>>>'. Na parte final, alguns tópicos sobre *Deep Learning* serão discutidos, incluindo o modelo de língua BERT [Devlin et al. 2019].

De maneira geral, a tarefa de **classificação** consiste em atribuir o rótulo correto para uma determinada entrada. Por exemplo, definir se o texto de uma notícia é ou não irônico, para o caso de *classificação binária*. Ressalta-se que em tarefas básicas, cada entrada é considerada isoladamente das outras, e o conjunto de rótulos é definido com antecedência. Ademais, a tarefa de classificação tem algumas variantes interessantes,

[3]https://spacy.io/

por exemplo, na *classificação multiclasse*, cada instância pode receber vários rótulos; na *classificação de classe aberta*, o conjunto de rótulos não é definido com antecedência; e na *classificação de sequência*, uma lista de entradas é classificada em conjunto.

Adicionalmente, um classificador é denominado supervisionado se for construído com base em conjunto de dados de treinamento contendo o rótulo correto para cada entrada. A estrutura usada pela classificação supervisionada é mostrada na Figura 1.1, com as etapas de treinamento e predição.

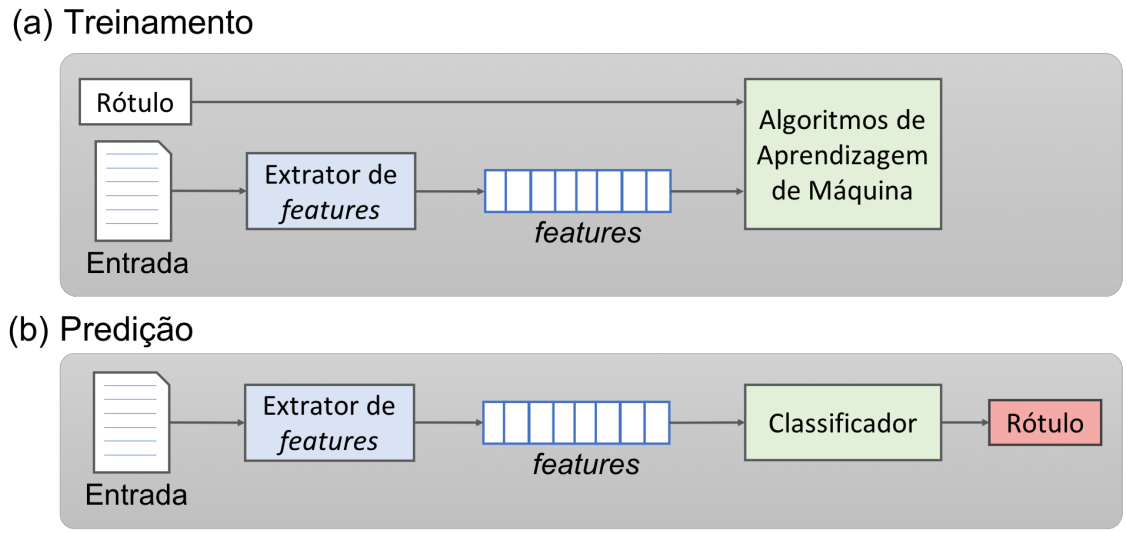


Figura 1.1: Classificação supervisionada. Adaptado de [Bird et al. 2009].

Observando a figura, percebe-se que durante a etapa de treinamento, um extrator de *features* é usado para converter cada texto de entrada em um vetor de *features*. Essas *features* devem capturar as informações básicas sobre cada texto e são usadas para classificá-lo. Já na etapa de predição, o mesmo extrator de *features* é utilizado para converter um novo texto no vetor de *features* correspondente e, em seguida, alimentar o classificador para prever o rótulo associado.

Além desta seção introdutória, o restante do minicurso será organizado em cinco seções: a seção 1.2 apresenta os principais conceitos e recursos usados na área de PLN, incluindo *Corpora* anotados e léxicos de várias tarefas, bem como ferramentas para análise de textos, como: *Part-Of-Speech Taggers* e *Stemmers*; a seção 1.3 discute modelos tradicionais usados no processo de classificação de textos; a seção 1.4 descreve brevemente dois modelos de *Word Embeddings*; a seção 1.5 apresenta modelos de *Deep Learning*, incluindo as redes neurais profundas e os modelos de língua (BERT); e, finalmente, a seção 1.6 conclui o trabalho e aponta perspectivas de trabalhos futuros.

Recomenda-se ainda três livros textos que podem ser usados como bibliografia básica em cursos da área: i) Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit [Bird et al. 2009], dos autores Steven Bird, Ewan Klein, and Edward Loper; ii) Inteligência Artificial - Uma Abordagem de Aprendizado de Máquina [Faceli et al. 2021], dos autores Katti Faceli, Ana Carolina Lorena, João Gama, Tiago Agostinho de Almeida e André C. P. L. F de Carvalho; e iii) Neural Network

Methods for Natural Language Processing [Goldberg 2017], do autor Yoav Goldberg.

Por fim, o procedimento de instalação das bibliotecas Python sugeridas está fora do escopo deste trabalho. No entanto, indicamos as seguintes urls `https://www.nltk.org/install.html`, `https://spacy.io/usage#installation` e `https://scikit-learn.org/stable/install.html`.

1.2. Recursos Léxicos

1.2.1. Conceitos e Definições

Esta subseção apresenta alguns termos, conceitos e definições amplamente utilizados nas diversas aplicações da área de PLN.

- **Corpus (plural: "Corpora"):** Um conjunto de dados linguísticos, sistematizados segundo determinados critérios, suficientemente extensos em amplitude e profundidade, de maneira que sejam representativos da totalidade do uso linguístico ou de algum de seus âmbitos, dispostos de tal modo que possam ser processados por computador. [Sanchez,1995] apud [Sardinha 2000];

- **Tokenização:** É o processo de separar um texto em *tokens* (palavras, números e símbolos). Alguns tokenizadores usam espaços, tabulações e quebras de linhas como separadores, enquanto que outros utilizam também os sinais de pontuação;

- **Normalização:** É o processo utilizado para garantir uma padronização mínima dos dados a serem processados. Por exemplo, conversão de todas as letras para minúsculas, correção de erros gramaticais comuns, remoção de imagens, URLs, *hashtags*, citações, espaços em excesso e pontuação duplicada, remoção de *stopwords* e letras repetidas, entre outros;

- **Stopwords:** São palavras que podem ser consideradas irrelevantes para o entendimento de um texto, ou seja, são palavras que geralmente têm pouco conteúdo lexical. Por exemplo, artigos, pronomes e preposições.

- **Stemmer:** É uma ferramenta utilizada para reduzir um termo ao seu radical através da remoção de desinências, afixos e vogais temáticas. *Stemming* é considerado um processo de normalização de texto;

- **Lematizer:** É uma ferramenta usada para agrupar diferentes inflexões e variantes de um termo para que possam ser analisadas como um único item, o lema. A lematização é uma operação mais poderosa e leva em consideração a análise morfológica das palavras, sendo que o lema deve obrigatoriamente existir em um dicionário;

- **Etiquetador (POS Tagger):** É uma ferramenta usada para atribuir a classe gramatical a cada uma das palavras de um texto, baseado tanto na sua definição quanto em seu contexto (palavras adjacentes) [Jurafsky and Martin 2009];

- **Analisador Sintático (Parser):** É uma ferramenta usada para analisar os modos de combinação de regras gramaticais (i.e., a função da palavra na oração), com a finalidade de gerar uma árvore que represente a estrutura sintática da oração analisada.

1.2.2. Corpora no NLTK

O NLTK disponibiliza diversos *Corpora* e coleções de livros, que podem ser baixados para a sua máquina, logo após a instalação do NLTK. Para mais informações, veja o capítulo 1 do livro NLTK [Bird et al. 2009], disponível em `https://www.nltk.org/book/ch01.html`. Aqui, discutiremos dois *Corpora* normalmente usados em aplicações para a língua portuguesa.

1.2.2.1. MacMorpho Corpus

Este *Corpus* foi desenvolvido pelo NILC/USP e contém 1 milhão de palavras etiquetadas com a classe gramatical. Normalmente, ele é usado para treinar etiquetadores (*POS Tagger*) para serem utilizados em aplicações. O código Python abaixo é usado para carregar o MacMorpho em memória e imprimir as palavras iniciais do *Corpus* com as respectivas etiquetas, sendo a etiqueta (tag) 'N' para substantivo e a tag 'V' para verbo.

```
from nltk.corpus import mac_morpho
print(mac_morpho.tagged_words())
>>> [('Jersei',  'N'), ('atinge', 'V'), ('média', 'N'), ...]
```

1.2.2.2. Stopwords Corpus

Este *Corpus* contém 2400 palavras consideradas *stopwords* em 11 línguas diferentes. O código abaixo é usado para carregar o recurso para a língua portuguesa em memória e comando *print(sw[0:30])* mostra as trinta primeiras palavras.

```
from nltk.corpus import stopwords
sw = stopwords.words('portuguese')
print(sw[0:30])
>>> ['de','a','o','que','e','é','do','da','em','um','para',
'com','não','uma','os','no','se','na','por','mais','as',
'dos','como','mas','ao','ele','das','à','seu','sua']
```

1.2.2.3. Carregando o seu próprio Corpus

O NLTK possui diversos métodos que podem ser usados para acessar informações de um *Corpus*. A Tabela 1.1 mostra algumas dessas funcionalidades.

A classe *PlaintextCorpusReader* é usada para acessar os métodos citados acima, considerando uma coleção de textos quaisquer. O código abaixo é usado para carregar os arquivos '.txt' existentes na pasta './corpus/'. O resultado da execução do trecho de código mostra que existem dois arquivos na pasta, além das palavras iniciais do *Corpus*.

```
from nltk.corpus import PlaintextCorpusReader
local = "./corpus/"
```

Tabela 1.1: Funcionalidades básicas do NLTK para acessar *Corpora*.

Métodos	**Descrição**
fileids()	os arquivos do *Corpus*
categories()	as categorias no caso de *Corpora* categorizados
words()	as palavras de todo o *Corpus*
words(fileid=[f1,f2,f3])	as palavras dos arquivos especificados
words(categories=[c1,c2,c3])	as palavras das categorias especificadas
sents()	as sentenças de todo o *Corpus*

```
corpusIronia = PlaintextCorpusReader(local,".*\.txt")
print(corpusIronia.fileids())
print(corpusIronia.words())
>>> ['tweetsIronia.txt', 'tweetsNaoIronia.txt']
>>> ['Quando', 'cheguei', 'a', 'Montemor', ',', 'gozavam',
 ↪  ...]
```

Existem outras formas de trabalhar com dados textuais na linguagem Python, com destaque para o uso da biblioteca Pandas[4]. O código abaixo é usado para carregar as informações de *tweets* sobre ironia, disponíveis no arquivo 'ironia.csv'. O comando *dados.info()* retorna as colunas de um registro do arquivo e o comando *dados['text']* retorna as informações do campo 'text' dos registros iniciais e finais.

```
import pandas as pd
file = "./corpus/ironia.csv"
dados = pd.read_csv(file, encoding='utf-8', decimal='.',
    sep=';', error_bad_lines=False)
print(dados.info())
print(dados['text'])
```

1.2.3. Etiquetadores

O NLTK possui quatros classes de etiquetadores (POS Taggers) que podem ser usados em aplicações de PLN, a saber: *DefaultTagger*, *UnigramTagger*, *BigramTagger* e *TrigramTagger*. A classe *DefaultTagger* atribui uma única etiqueta para todas as palavras. A classe *UnigramTagger* treina o modelo considerando as etiquetas de palavras únicas, enquanto as classes *BigramTagger* e *TrigramTagger* treinam os modelos considerando as etiquetas de sequências de duas e três palavras, respectivamente. O código abaixo é usado para treinar e testar três etiquetadores, usando as sentenças do *Corpus Mac_Morpho*, na proporção de 90%/10%. A parte final do código mostra-se um exemplo de etiquetagem de uma sentença, usando o *BigramTagger* que obteve 85,5% de acurácia.

```
import nltk
from nltk.corpus import mac_morpho
```

[4]https://pandas.pydata.org/docs/index.html#

```
# 10% para teste e 90% para treino
prop = int(0.1 * len(mac_morpho.tagged_sents()))
treino = mac_morpho.tagged_sents()[prop:]
teste = mac_morpho.tagged_sents()[:prop]
# Etiquetador Padrão
etiq1 = nltk.DefaultTagger('N')
print("BASIC Tagger: ", etiq1.evaluate(teste))
#Etiquetador UNIGRAM
etiq2 = nltk.UnigramTagger(treino, backoff=etiq1)
print("UNIGRAM Tagger: ", etiq2.evaluate(teste))
#Etiquetador BIGRAM
etiq3 = nltk.BigramTagger(treino, backoff=etiq2)
print("BIGRAM Tagger: ", etiq3.evaluate(teste))
frase = nltk.word_tokenize("O governo não pode dar educação
 ↪  porque a educação derruba o governo")
print(etiq3.tag(frase), end='\n')
>>> BASIC Tagger:  0.2079790656025594
>>> UNIGRAM Tagger:  0.837488915549528
>>> BIGRAM Tagger:  0.8548849825256899
>>> [('O', 'ART'), ('governo', 'N'), ('não', 'ADV'),
 ('pode', 'VAUX'), ('dar', 'V'), ('educação', 'N'),
 ('porque', 'KS'), ('a', 'ART'), ('educação', 'N'),
 ('derruba', 'V'), ('o', 'ART'), ('governo', 'N')]
```

1.2.4. Normalizadores

O processo de normalização de textos pode ser entendido como alguns passos de pre-processamento, incluindo a correção de erros gramaticais, transformação do texto para letras minúsculas, remoção de letras e/ou palavras duplicadas, remoção de acentos e *stopwords*, uso de *stemming* e/ou *lemas*, entre outros.

Os erros gramaticais podem ser identificados através do uso do corretor gramatical CoGrOO [Kinoshita et al. 2007], que verifica a colocação pronominal, concordância nominal e verbal, concordância entre sujeito e verbo, uso de crase, regência nominal e verbal, além de outros erros comuns da língua portuguesa escrita.

Com relação a *Stemmers*, o NLTK inclui diversos algoritmos facilmente disponíveis, por exemplo o Porter e Lancaster stemmers seguem suas próprias regras para remover afixos. O código abaixo é usado para instanciar e aplicar esses *stemmers* em uma determinada frase. Neste exemplo, a versão do *Porter Stemmer* utilizada é para a língua inglesa e, praticamente, não atua em palavras da língua portuguesa. Porém, existe o projeto *PTStemmer - A stemming toolkit for Portuguese in Python*[5], que implementa versões dos *Stemmers* Orengo, Porter e Savoy.

```
import nltk
texto="O governo não pode dar educação porque a educação
 ↪  derruba o governo."
```

[5]https://github.com/lsdr/ptstemmer

```
tokens = nltk.word_tokenize(texto)
porter = nltk.PorterStemmer()
rslp = nltk.stem.RSLPStemmer()
outPorter=[]
outRSLP=[]
for tok in tokens:
    outPorter.append(porter.stem(tok))
    outRSLP.append(rslp.stem(tok))
print(outPorter, '\n', outRSLP)
>>> ['O', 'governo', 'não', 'pode', 'dar', 'educação',
 'porqu', 'a', 'educação', 'derruba', 'o', 'governo']
>>> ['o', 'govern', 'não', 'pod', 'dar', 'educ', 'porqu',
 'a', 'educ', 'derrub', 'o', 'govern']
```

1.2.5. Analisadores Sintáticos (*Parsing*)

A classe *RegexpParser* do pacote *nltk.chunk* permite explorar estruturas gramaticais de textos. Além dessa classe, o NLTK possui diversas outras para tratar gramáticas livres de contexto (GLC)[6]. O código abaixo mostra uma gramática relativamente simples para identificar sintagmas nominais (SN) em textos, composto de um artigo seguindo de substantivo. Destaca-se que o etiquetador (*etiq3*) e o objeto *tokens* são os mesmos definidos nos exemplos anteriores.

```
from nltk.chunk import RegexpParser
tags = etiq3.tag(tokens)
analiseGramatical = RegexpParser(r"""
    SN:{<ART><N>}
    }<V>{
    """)
arvore = analiseGramatical.parse(tags)
arvore.draw()
```

Como resultado da execução do código tem-se a Figura 1.2. Assim, com o uso de gramáticas é possível encontrar elementos textuais nas descrições. Essa técnica é conhecida na literatura como identificação de padrões linguísticos, e tem sido explorada com sucesso na área de análise de sentimentos e mineração de opiniões sobre produtos ou serviços. [Anchiêta et al. 2013] usaram alguns padrões para identificar atributos e métodos em descrições de casos de uso da UML e [Sousa 2015] também utilizou padrões linguísticos para identificar aspectos de um produto e as palavras opinativas sobre tais aspectos. Na gramática, ele utilizou substantivos, adjetivos, verbos e advérbios para identificar esses elementos.

[6]GLC é uma gramática formal onde todas as regras de produções são da forma $A \rightarrow \alpha$, sendo A um símbolo não-terminal e α um regra de produção composta por terminais e não-terminais

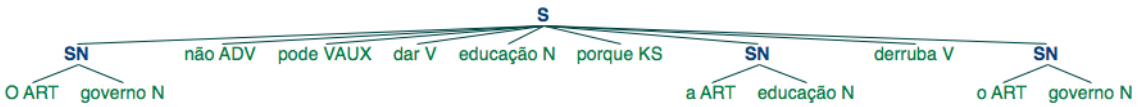


Figura 1.2: Árvore sintática parcial.

1.2.6. Outros Recursos

Existem algumas alternativas à biblioteca NLTK, como, por exemplo, as ferramentas spaCy[7], nlpnet [Fonseca and Rosa 2013], stanza [Qi et al. 2020], NLPyPort [Ferreira et al. 2019], entre outras. Aqui, focaremos na spaCy, pois ela é uma ferramenta robusta com modelos treinados para várias tarefas de PLN.

O spaCy é uma ferramenta que tem suporte a mais de 60 línguas, possuindo modelos treinados para tarefas de reconhecimento de entidades nomeadas, etiquetação morfossintática (*tagger*), análise de dependência (*dependency parsing*), lematização, entre outras. Para o português, o spaCy possui três modelos treinados que variam em tamanho e acurácia, quanto menor o modelo, menor a acurácia, são eles: `pt_core_news_sm`, `pt_core_news_md` e `pt_core_news_lg`, onde o primeiro é o menor e último o maior. Para mais detalhes sobre esses modelos, sugere-se a páginal oficial da ferramenta[8].

Um dos usos mais comuns da ferramenta é a extração de informações morfológicas, morfossintáticas e sintáticas de um texto. O código em Python abaixo ilustra um exemplo dessa tarefa. Neste exemplo, `token.text` é o texto original da sentença, `token.lemma_` é a forma canônica da palavra, `token.morph` é a informação morfológica da palavra `token.pos_` é a etiqueta morfossintática da palavra, `token.dep_` é a relação de dependência entre as palavras, `token.is_alpha` informa se a palavra é um texto e `token.is_stop` indica se a palavra é uma *stopword*.

```
import spacy

nlp = spacy.load("pt_core_news_sm")  # modelo
doc =  nlp("Justiça aceita denúncia")  # sentença
for token in doc:
    print(token.text, token.lemma_, token.morph,
     ↪  token.pos_, token.dep_, token.is_alpha,
     ↪  token.is_stop)

>>> "Justiça", "Justiça", "Gender=Fem|Number=Sing", "NOUN",
 ↪  "nsubj", "True", "False"
>>> "aceita", "aceito",
 ↪  "Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin",
 ↪  "VERB", "ROOT", "True", "False"
>>> "denúncia", "denúncia", "Gender=Fem|Number=Sing",
 ↪  "NOUN", "obj", "True", "False"
```

[7] https://spacy.io/
[8] https://spacy.io/models/pt

Outro uso comum da ferramenta spaCy é para a tarefa reconhecimento de entidades nomeadas cujo objetivo é identificar entidades nomeadas dentro e um conjunto de categorias pré-definidas, tais como: Pessoa, Localização, Organização, entre outras. O código abaixo exemplifica essa tarefa.

```
import spacy

nlp = spacy.load("pt_core_news_sm")   # modelo
doc =  nlp("Quixadá é uma bela cidade.")   # sentença
for ent in doc.ents:
    print(ent.text, ent.label_)

>>> 'Quixadá', 'LOC'
```

No exemplo, `ent.text` é o texto original e `ent.label_` é o rótulo da entidade reconhecida. Aqui, o spaCy identificou que a palavra `Quixadá` é uma localização.

Além dos exemplos acima, o spaCy permite criar regras a fim de melhorar ou adaptar o resultado obtido pelos modelos. No que segue, serão apresentados alguns algoritmos tradicionais de aprendizado de máquina aplicados em tarefas de PLN.

1.3. Classificação: Algoritmos tradicionais

A tarefa de **classificação de Textos** consiste em atribuir o rótulo de classe para uma determinada entrada. Por exemplo, definir se o texto de um *tweet* ou notícia é ou não irônico; se uma mensagem possui ou não discurso de ódio; decidir qual é o tópico de um artigo de notícias (esporte, tecnologia, política, ...), entre outros.

A seguir, apresenta-se um classificador para determinar se o texto de um *tweet* é ou não irônico, baseado no modelo mostrado na Figura 1.1. Os dados sobre os *tweets* irônicos e não irônicos foram disponibilizados pelos organizadores da tarefa IDPT (*Irony Detection in Portuguese*), que faz parte do evento IberLEF 2021 (*Iberian Languages Evaluation Forum*). A partir desses dados organizou-se dois arquivos: 'tweetsIronia.txt' e 'tweetsNaoIronia.txt', contendo 12.807 e 2.476 mensagens, respectivamente.

1.3.1. Classificador de *Tweets* Irônicos

O desenvolvimento de um classificador pode ser dividido nos seguintes passos:

1. **Carregar dados:** Inicialmente, deve-se carregar os dados na memória, atribuindo o rótulo de cada classe. Na parte final do código, o comando 'y, text = zip(*dados)' permite separar as classes e os textos nos objetos 'y' e 'text', respectivamente:

   ```
   dados = []
   with open('db/tweetsIronia.txt') as f:
       d = [l.strip().split('\n') for l in f.readlines()]
   for t in d:
       dados.append(['1']+t)
   ```

```
with open('db/tweetsNaoIronia.txt') as f:
    d = [l.strip().split('\n') for l in f.readlines()]
for t in data:
    dados.append(['0']+t)

y, text = zip(*dados)
print(collections.Counter(y))
>>> Counter({'1': 12807, '0': 2476})
```

2. **Definir os conjuntos de treino e teste:** O segundo passo é separar os dados nos conjuntos de treinamento e teste. Isso é feito através do método 'train_test_split' da biblioteca Sklearn, conforme comandos abaixo. No caso, a proporção utilizada para treinar e testar os dados foi de (75% / 25%). O parâmetro 'stratify' permite que os dados sejam divididos de forma estratificada, considerando os rótulos da classe:

```
text_train, text_test, y_train, y_test =
 ↪  train_test_split(text, y, random_state=1,
 ↪  test_size=0.25, stratify=y)
```

3. **Vetorização dos dados:** As bibliotecas de aprendizagem de máquina esperam entrada na forma de arrays de floats, com dimensão fixa. Então, é necessário fazer a conversão dos textos em tais representações. Essa forma de representação é chamada *Bag of Words*, em que um texto é transformado em uma lista de tokens e, posteriormente, no array de floats, considerando o vocabulário dos textos. Os seguintes comandos fazem a conversão dos textos para arrays de floats, usando o método 'TfidfVectorizer'. No entanto, existem outros métodos de vetorização, a saber: *HashingVectorizer* e *CountVectorizer*. O método *TfidfVectorizer* considera os tokens como palavras (words), elimina as *stopwords* presentes no objeto 'sw' e usa no máximo 1500 *features* como dimensão do array.

```
sw = stopwords.words('portuguese')
vet = TfidfVectorizer(analyzer="word", stop_words=sw,
 ↪  max_features=1500)
vet.fit(text_train)
X_train = vet.transform(text_train)
X_test = vet.transform(text_test)
print("Treino/Teste: ", X_train.shape, X_test.shape)
>>> Treino/Teste: (11462, 1500) (3821, 1500)
```

No exemplo, os objetos 'X_train' e 'X_test' são do tipo *'scipy.sparse.csr.csr_matrix'*, chamada *matriz documento-termo*, que é uma matriz esparsa da forma (nrAmostras, nrFeatures). Destaca-se que o conjunto de *features* representa o vocabulário dos dados e pode ser visualizado através do comando `print(vet.vocabulary_)`.

4. **Treinar o classificador:** A biblioteca *Sklearn* possui diversos algoritmos tradicionais para a tarefa de classificação, incluindo as máquinas de vetores de suporte

(do inglês: *Support Vector Machines - SVM*), *Naïve Bayes* e as árvores de decisão (do inglês: *Decision Tree*). A fundamentação matemática para o entendimento dos algoritmos está fora do escopo deste trabalho. Aqui, vamos utilizar o método 'SVC', cuja implementação é baseada no algoritmo 'libsvm'. Os comandos a seguir são usados para instanciar e treinar esse algoritmo. Para conhecimento geral dos algoritmos disponíveis na biblioteca *Sklearn*, sugere-se a documentação oficial [9].

```
svc = SVC()
svc.fit(X_train, y_train)
```

5. **Avaliar o modelo:** O passo final do processo é avaliar o desempenho do algoritmo. Existem várias métricas que são usadas para avaliar as fases de treinamento e teste dos classificadores. Quando as classes são balanceadas, normalmente utiliza-se a Acurácia, que consiste na razão entre o número de predições verdadeiras e o número total de amostras. Para problemas de classificação binária e multiclasse, a matriz de confusão é utilizada para identificar a performance do modelo para cada classe. A matriz fornece quantas amostras de cada classe foram classificadas corretamente e quantas foram confundidas com outras classes. O código abaixo mostra a Acurácia do modelo para os dados de treino e teste, além de apresentar a matriz de confusão do modelo para os dados de teste.

```
print("Acc (Treino): %.2f"%svc.score(X_train, y_train))
print("Acc (Teste): %.2f"%svc.score(X_test, y_test))
print("Matriz de Confusão:")
print(confusion_matrix(y_test, svc.predict(X_test)))
>>> Acc (Treino): 0.98
>>> Acc (Teste): 0.96
>>> Matriz de Confusão:
>>> [[ 480  139]
     [  31 3171]]
```

Observa-se que, a Acurácia do modelo foi de 98% para os dados de treinamento e 96% para os dados de teste. A matriz de confusão segue a seguinte forma:

```
[[ TN, FP]
 [ FN, TP]]
```

sendo:

- **TN (*True Negative*):** A quantidade de rótulos da classe negativa que o modelo previu corretamente como negativo.
- **FP (*False Positive*):** A quantidade de rótulos que originalmente pertencem a classe negativa, mas o modelo previu como positivo.
- **FN (*False Negative*):** A quantidade de rótulos que originalmente pertencem a classe positiva, mas o modelo previu como negativo.

[9] https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

- **TP (*True Positive*):** A quantidade de rótulos da classe positiva que o modelo previu corretamente como positivo.

A biblioteca *Sklearn* possui também um relatório de classificação que fornece as métricas de **precisão, cobertura, medida-f e a quantidade de amostras de cada classe (*support*)**. Essas medidas são definidas da seguinte forma:

- **Precisão (P):** Número de amostras de uma determinada classe que são realmente daquela classe. $P = TP/(TP+FP)$.
- **Cobertura (Recall ou R):** Número de predições corretas de uma determinada classe. $R = TP/(TP+FN)$.
- **Medida-F (F1-Score ou F1):** É a média harmônica entre a precisão e a cobertura. $F1 = 2*(P*R)/(P+R)$.

O código abaixo é utilizado para imprimir o relatório da classificação do exemplo em questão e a Figura 1.3 mostra o resultado da precisão, cobertura e medida-f de cada classe considerando os dados do conjunto de teste. O relatório mostra que a medida-f da classe de ironia foi de 97% e para a classe não ironia foi inferior (apenas de 85%). Isto aconteceu devido a baixa cobertura desta segunda classe.

```
print("Relatorio de Classificacao")
print(classification_report(y_test,
 ↪  svc.predict(X_test)))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.94      0.78      0.85       619
           1       0.96      0.99      0.97      3202

    accuracy                           0.96      3821
   macro avg       0.95      0.88      0.91      3821
weighted avg       0.95      0.96      0.95      3821
```

Figura 1.3: Saída do relatório de classificação.

Para o caso de classe desbalanceadas, a curva ROC (do inglês: *Receiver Operating Characteristic*) ajuda a entender a performance do modelo. A Curva ROC funciona com a saída da função de predição, definindo diferentes valores (*threshold*) para descobrir diferentes taxas de falsos positivos (FP) e verdadeiros positivos (TP) de acordo com o *threshold*. O código a seguir é usado para plotar a curva ROC para o classificador SVC, considerando o conjunto de teste utilizado no exemplo. A Figura 1.4 mostra a representação gráfica resultante.

```
df = svc.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, df,
 ↪  pos_label='1')
acc = svc.score(X_test, y_test)
auc = roc_auc_score(y_test,
 ↪  svc.decision_function(X_test))
```

```
with plt.style.context(('ggplot','seaborn')):
    plt.figure(figsize=(8,6))
    plt.scatter(fpr, tpr, c='blue')
    plt.plot(fpr, tpr, label="Acuracia:%.2f AUC:%.2f" %
     ↪  (acc, auc), linewidth=2, c='red')
    plt.xlabel("Taxa FP")
    plt.ylabel("Taxa TP")
    plt.title('Curva ROC')
    plt.legend(loc='best');
```

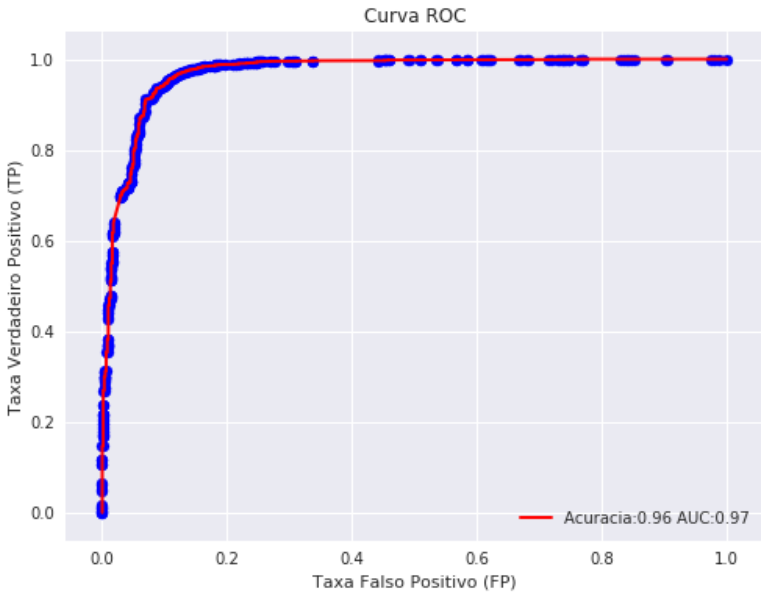


Figura 1.4: Curva ROC

Destaca-se que uma curva ROC é uma demonstração bidimensional da performance de um classificador. Para comparar classificadores é preciso reduzir a curva ROC a um valor escalar. Normalmente, usa-se o valor da área abaixo da curva (do inglês: *Area Under the Curve - AUC*). No exemplo, o valor AUC foi de 97%.

1.4. Word Embeddings

Embora as técnicas de vetorização de palavras descritas anteriormente (*bag of words*, *TF-IDF*) se mostrem simples e eficientes, elas possuem sérias limitações. Conforme [Mikolov et al. 2013a], quando aplicadas em grandes volumes de dados estes modelos apresentam matrizes de alta dimensão, que afetam o custo computacional ao serem processadas. Aliado a este fator, é comum encontrar a característica de esparsidade, que é a ausência do valor do grau de importância de uma determinada célula da matriz devido ao fato daquele termo não estar presente no documento associado. Em determinados documentos algumas palavras podem não estar presentes, e desta forma na matriz *termo x documento* o respectivo grau de importância é representado com o valor '0'. Por fim,

outra importante desvantagem é ausência de informações semânticas pois estes modelos ignoram o contexto e a ordem das palavras nos documentos [Manning et al. 2008].

Diferente dos modelos clássicos de Recuperação da Informação, as *Word Embeddings* representam cada uma das palavras como um vetor n-dimensional contínuo de números reais, de forma que esta representação é capaz de capturar relações de sintáticas e semânticas (e.g. similaridades). Essas relações são derivadas das posições destas palavras em um espaço vetorial, i.e., se estas palavras estão em regiões próximas, é possível computar a distância entre os vetores (*e.g.,* distância cosseno) e encontrar relações de similaridade entres elas[Mikolov et al. 2013b]. A Figura 1.5 apresenta uma abstração de um espaço vetorial onde é possível verificar a proximidade de palavras similares.

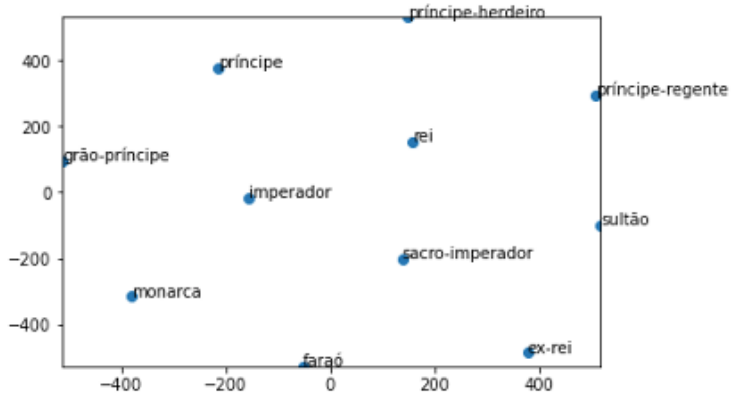


Figura 1.5: Representação abstrata de espaço vetorial: algoritmo Word2Vec

Como é possível verificar na figura, palavras como "rei", "sultão", "imperador"e "faraó"estão em regiões próximas no espaço vetorial, o que demonstra a característica de similaridade pois são termos sinônimos. Tais características são usadas para analisar diversas tarefas de PLN, como análise de sentimentos e detecção de discurso de ódio.

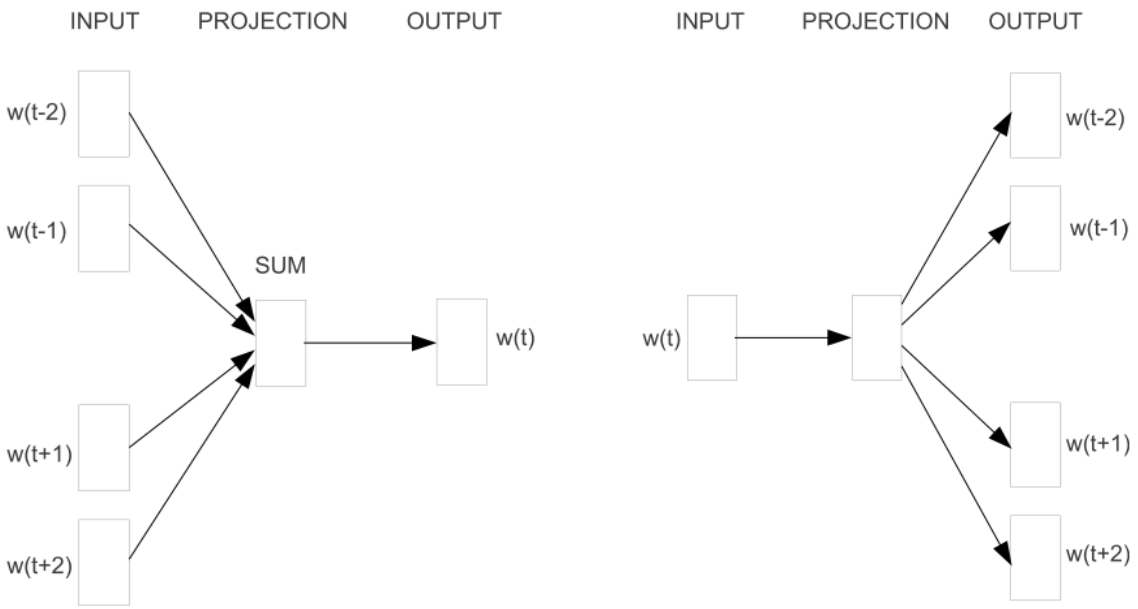
Alguns dos algoritmos mais utilizados na literatura de *Word Embeddings* são *Word2Vec* e *Paragraph Vector* (ou *Doc2Vec*). Ambos algoritmos são baseados em Redes Neurais Artificiais de duas camadas(camada de projeção e camada de saída), onde a partir de um *corpus* de entrada, ao gerarem vetores de palavras para cada um dos termos, buscam mapear estes vetores em um espaço vetorial onde é possível representar as palavras similares em vetores similares no espaço vetorial [Mikolov et al. 2013a].

1.4.1. Word2Vec

Conforme apresentado em [Mikolov et al. 2013a] [Mikolov et al. 2013b], o algoritmo *Word2vec* possui duas variações para o treinamento, são elas *Continous Bag of Words* (CBOW) e *Skip gram*. No algoritmo CBOW, a predição do termo central é dada a partir uma janela de palavras em um determinado contexto. A entrada da Rede Neural são as palavras (juntamente com seu vetor numérico) em uma dada janela, a camada oculta representa o número de dimensões na qual é desejado representar o termo central, que é o elemento da camada de saída. A Figura 1.6a apresenta uma ilustração deste algoritmo.

O funcionamento do algoritmo *Skip gram* é o oposto do CBOW. A partir de um

termo central, esta técnica busca predizer um conjunto de vetores de palavras dada uma janela de termos. Neste caso a camada de entrada é o termo central, a camada oculta também contém o número de dimensões na qual é desejado representar o termo central, e por fim a camada de saída apresenta o conjunto de termos em torno da palavra central. A Figura 1.6b mostra um exemplo deste algoritmo.



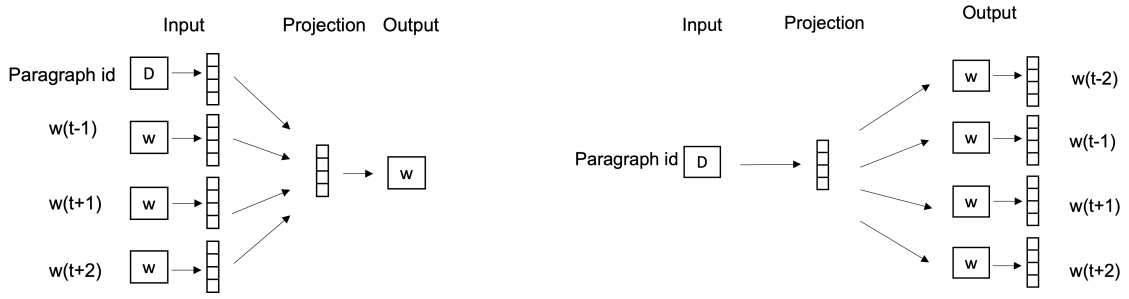
(a) Método CBOW (b) Método Skip-Gram

Figura 1.6: Algortimo Word2Vec: Variações para treinamento

1.4.2. Doc2vec

De acordo com [Le and Mikolov 2014], *Paragraph Vector* é um método não supervisionado que busca aprender um tamanho fixo de características a partir de diferentes formas textuais e.g., senteças, parágrafos ou documentos. Os autores apresentam duas abordagem deste algoritmo, *Distributed Memory* (DM) e *Distributed Bag of Words* (DBOW). O método DM, utiliza vetores de palavras junto a representação do vetor do parágrafo (sentença, ou documento) para a predição de um determinado termo dentro da sentença. Esta representação do vetor parágrafo atua como uma espécie de memória, sendo capaz de conceder informações contextuais à tarefa de predição. A Figura 1.7a apresenta o esquema do método DM. É possível perceber a semelhança ao método CBOW (Figura 1.6a), onde a diferença consiste no vetor *Paragraph Id* que é compartilhado pelos termos daquele contexto que está sendo utilizado no treinamento.

Ao contrário do método DM, a técnica DBOW utiliza somente as informações presentes no *Paragraph Id* (ver Figura 1.7b). Este método faz uma seleção aleatória de palavras na representação do vetor de parágrafos e usa estas informações para prever o termo central. Este método já se assemelha ao *Skip gram* (ver Figura 1.6b).



(a) Método DM (b) Método DBOW

Figura 1.7: Algortimo Paragraph Vector.

1.5. Algoritmos de Deep Learning

Apesar dos interessantes resultados alcançados pelos algoritmos tradicionais de aprendizado de máquina, eles sofrem com a dificuldade de cobrir todas as regularidades de uma língua através do desenvolvimento manual de *features* [Deng and Liu 2018]. Nesse contexto, os algoritmos de aprendizagem profunda (*deep learning)* são usados para preencher essa lacuna. *Deep learning* pode ser definido como um classe de técnicas de aprendizado de máquina que explora muitas camadas de processamento de informação não linear para extração e transformação automática de *features* para análise e classificação de padrões [Deng and Yu 2014]. Essa classe de algoritmos é baseada nas Redes Neurais e popularizou-se devido aos avanços tanto em *hardware* quanto em *software* e, principalmente, por causa dos resultados alcançados em diversas áreas. Aqui, serão apresentados dois tipos de algoritmos de aprendizagem profunda, Redes Convolucionais na subseção 1.5.1 e Redes Recorrentes na subseção 1.5.2.

1.5.1. Redes Convolucionais

As Redes Neurais Convolucionais (do inglês: *Convolutional Neural Networks - CNN*) foram introduzidas por [LeCun 1989] e são um tipo especializado de rede neural para processar dados que possuem uma topologia semelhante a uma grade, como as imagens.

Para a área de PLN essas redes foram inicialmente usadas para tarefas de anotação de papéis semânticos [Collobert et al. 2011], análise de sentimentos [Kim 2014] e classificação de questões [Kalchbrenner et al. 2014].

As principais operações de uma rede convolucional são a **convolução** e o ***pooling***. A **convolução** pode ser entendida como aplicação de uma função não linear sobre cada instância de uma janela deslizante de k-palavras sobre as sentenças. Essa função, chamada de filtro, transforma uma janela de k-palavras em um vetor de dimensão l. Cada dimensão corresponde a um filtro que captura propriedades importantes das palavras na janela [Goldberg 2017]. A Figura 1.8 mostra um exemplo de convolução sobre o texto "*Quixadá é bonita*". Como pode ser observado, o filtro é aplicado sobre as *words embeddings* para produzir um mapa de *features* que representa um k-grama específico.

O objetivo de camada de ***pooling*** é focar nas *features* mais importantes de uma sentença. Existem diversas operações de *pooling*: *max-pooling*, *average pooling*, *k-max*

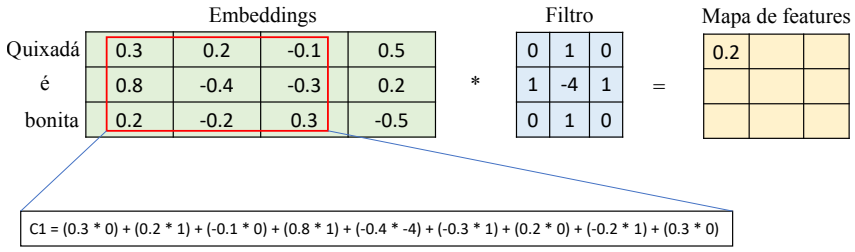


Figura 1.8: Exemplo de convolução sobre uma sentença.

pooling, entre outras [Goldberg 2017]. A mais comum é a *max-pooling* que pega o valor máximo em cada dimensão. Figura 1.9 exibe a operação de *max-pooling* para extrair os maiores valores da matriz, através de um filtro $2x2$ de passo 2.

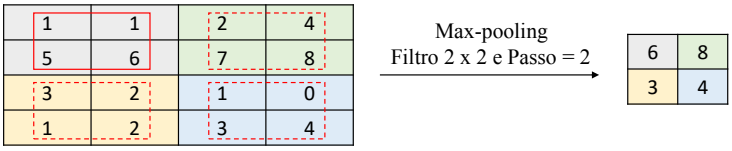


Figura 1.9: Exemplo da operação de max-pooling.

Pode-se perceber que a operação de *pooling* extraiu os maiores valores da matriz, baseado em um filtro $2x2$ de passo 2, resultando em uma matriz $2x2$. Diversos *frameworks* implementam essas operações, tais como: PyTorch[10], TensorFlow[11], Keras[12], entre outros. Esses *frameworks* são para a linguagem Python e eles abstraem as operações matemáticas por traz das redes neurais permitindo fazer uso das operações de convolução e *pooling* através de simples chamadas de métodos.

1.5.2. Redes Recorrentes

As Redes Recorrentes (do inglês: *Reccurent Neural Networks - RNNs*) foram introduzidas por [Rumelhart et al. 1986]. A principal característica dessa rede é trabalhar com dados sequenciais, como um texto. Assim como as redes convolucionais, as redes recorrentes são um tipo especializado de rede neural, no entanto elas são focadas em processar uma sequência de valores. Esse tipo de rede é bastante utilizado em tarefas de PLN, cuja entrada é uma sequência de texto e a saída também é uma sequência de texto, como a tradução automática.

De acordo com [Jurafsky and Martin 2009] uma rede recorrente é qualquer rede que contenha um ciclo em suas conexões de rede, ou seja, qualquer estrutura em que o valor de uma unidade seja direta ou indiretamente dependente de saídas anteriores como entrada. Na Figura 1.10, é exibido uma RNN que recebe uma entrada X no tempo t (X_t) e produz uma saída h_t. Essa saída e o próximo valor (X_{t+1}) são utilizados como entrada na próxima rede, produzindo a saída h_{t+1}. Esse processo continua até o final da sequência.

Tradicionalmente, uma rede recorrente percorre uma sequência da esquerda para a

[10] https://pytorch.org/
[11] https://www.tensorflow.org/
[12] https://keras.io/

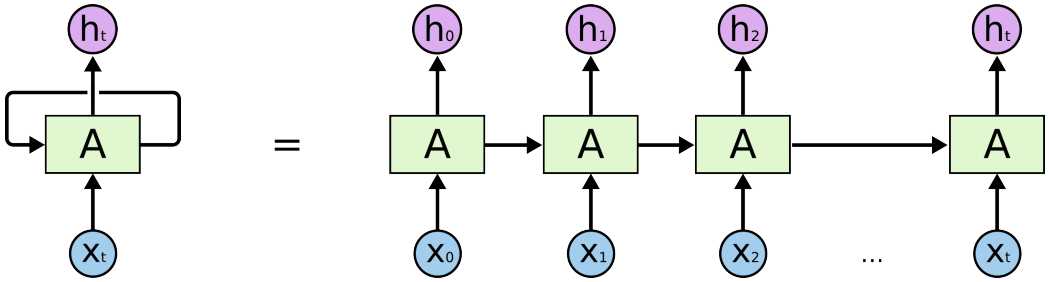


Figura 1.10: Estrutura de uma rede recorrente.

direita. No entanto, é possível percorrer de forma bidirecional: da esquerda para a direita e da direita para esquerda. Considerando a tarefa de etiquetagem gramatical, muitas vezes para se atribuir uma classe a uma palavra é necessário observar o contexto bidirecional da mesma. A Figura 1.11 exibe um exemplo desse tipo de rede.

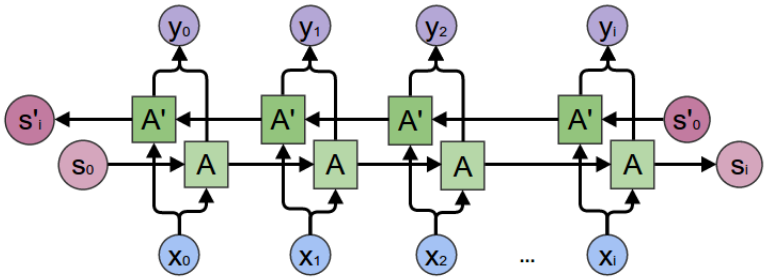


Figura 1.11: Estrutura de uma rede recorrente bidirecional.

A partir da figura acima, pode-se ver que o contexto está sendo analisado tanto de S_0 para S_i quanto de S'_0 para S'_i. Devido a capacidade de analisar o contexto de ambos os lados, as redes bidirecionais atingem resultados superiores as redes unidirecionais.

Além da bidirecionalidade, também é possível criar empilhar várias redes recorrentes, como apresentado na Figura 1.12. Nessa figura, a redes são unidirecionais, no entanto é possível criar redes recorrentes bidirecionais empilhadas. Nesse tipo de rede, a saída de cada unidade de processamento é a entrada para a rede recorrente no nível acima.

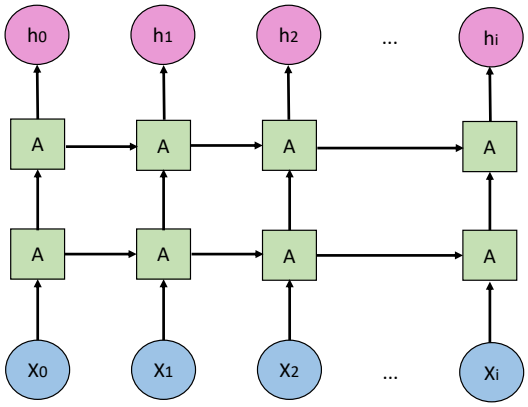


Figura 1.12: Exemplo de redes reccorentes empilhadas.

Existem dois tipos de redes recorrentes, *Long Short-Term Memory* (LSTM) e *Gated Reccurent Unit* (GRU). Essas redes foram criadas para resolver os problemas das RNNs convencionais que é o de "esquecer" informações ao longo do processamento em sentenças longas.

Assim como nas redes convolucionais, os *frameworks* PyTorch, TensorFlow e Keras também disponibilizam a implementação das redes recorrentes através de simples chamadas de métodos.

1.5.3. BERT

Bidirectional Encoder Representations from Transformer (BERT) [Devlin et al. 2019] é um modelo de língua baseado em *transformer*, que é um modelo de aprendizagem profunda que utiliza mecanismo de *self-attention* [Vaswani et al. 2017]. Modelo de língua é um modelo capaz de predizer a próxima palavra/sentença dado uma sequência prévia de palavras/sentenças. Por exemplo, a partir da sentença "*Tinha uma [MASK] no meio do caminho.*" um modelo de língua é treinado para predizer qual palavra deve substituir "[MASK]". Para esse exemplo, o BERT sugere que "[MASK]" deve ser substituída pela palavra "pedra". De modo simples, o mecanismo de atenção (*attention*) é um componente das redes profundas que é responsável em dar foco nos principais dados de entrada.

De forma geral, BERT é um modelo que utiliza vetores de sub-palavras mais *transformers*. Os vetores de sub-palavras possuem um contexto para cada vetor que comportam variações morfológicas diferentes das *embeddings* tradicionais. Os *transformers* não utilizam direcionalidade, ao invés disso eles utilizam o contexto inteiro da sentença de uma só vez. A arquitetura do BERT pode ser simplificada em duas partes, como exibido na Figura 1.13. No lado esquerdo, está o tokenizador que quebra as palavras em sub-palavras e o *encoder* responsável por produzir diferentes representações do texto de entrada. Ainda no lado esquerdo, o BERT é treinado em dados não rotulados, visando predizer a tag "[MASK]. Nessa tarefa, o BERT recebe um texto e, de forma aleatória, substitui algumas palavras por "[MASK]" a fim de posteriormente predizê-las. A partir desse treinamento em dados não rotulados, é possível fazer o treinamento em dados rotulados, ou seja, o *decoder* (lado direito). Esse treinamento é chamado de afinação (fine-tuning), pois o BERT vai ser ajustado para alguma tarefa específica, por exemplo, análise de sentimentos, tradução automática, sumarização automática, e assim por diante.

Essa capacidade de afinação e transferência de aprendizagem, uma vez que a partir do treinamento em dados não rotulados, O BERT é capaz é ser treinados em diversas tarefas de PLN e atingir resultados estado-da-arte é uma das grandes características desse modelo. Para mais detalhes sobre esse modelo de língua, sugere-se o seguinte guia ilustrado[13]. Os *frameworks* TensorFlow, PyTorch e keras possuem implementações do modelo para fácil uso, sendo a biblioteca HuggingFace[14] uma das mais famosas.

1.6. Conclusão

Conhecer várias técnicas de PLN é importante para desenvolver modelos de aprendizagem mais robustos e competitivos. Dessa forma, este capítulo apresentou uma visão geral

[13] `https://jalammar.github.io/illustrated-bert/`
[14] `https://github.com/huggingface/transformers`

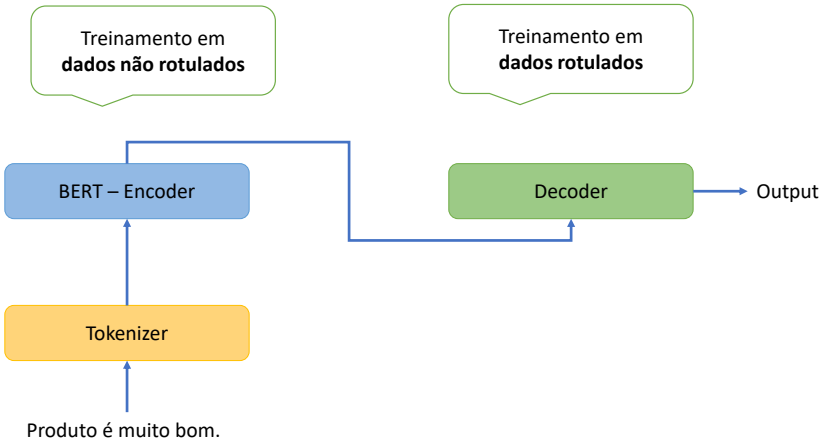


Figura 1.13: Arquitetura simplificada do modelo BERT.

sobre várias técnicas de Processamento de Língua Natural focadas na tarefa de classificação, iniciando com abordagens baseadas em *features*, *n*-gramas, léxicos passando por estratégias baseadas em *word embeddings* e finalizando em métodos baseados em aprendizado profundo e modelos língua.

A área de PLN tem focado principalmente em estratégias de aprendizado profundo e modelos de língua devido tanto a evolução do *software* quanto *hardware* e da disponibilidade de grandes *corpora*. A junção desses fatores proporcionou alcançar resultados impressionantes em diversas tarefas de PLN. Por fim, espera-se que este capítulo sirva como base e ajude no desenvolvimento de métodos cada vez mais robustos.

Referências

[Anchiêta et al. 2013] Anchiêta, R. T., de Sousa, R. F., and Moura, R. S. (2013). Using NLP techniques for identifying GUI prototypes and UML diagrams from use cases. In *Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering*, Boston, MA, USA.

[Anchiêta et al. 2015] Anchiêta, R. T., Ricarte Neto, F. A., de Sousa, R. F., and Moura, R. S. (2015). Using stylometric features for sentiment classification. In *Proceedings of the 16th International Conference on Intelligent Text Processing and Computational Processing*, Cairo, Egypt.

[Bird 2006] Bird, S. (2006). NLTK: The Natural Language Toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72, Sydney, Australia. Association for Computational Linguistics.

[Bird et al. 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly.

[Collobert et al. 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12.

[Deng and Liu 2018] Deng, L. and Liu, Y. (2018). *Deep Learning in Natural Language Processing*. Springer Singapore.

[Deng and Yu 2014] Deng, L. and Yu, D. (2014). Deep learning: methods and applications. *Foundations and trends in signal processing*, 7(3–4):197–387.

[Devlin et al. 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Minneapolis, Minnesota.

[Faceli et al. 2021] Faceli, K., Lorena, A. C., Gama, J., and Carvalho, A. C. P. L. F. D. (2021). *Inteligência artificial: uma abordagem de aprendizado de máquina*. LTC.

[Ferreira et al. 2019] Ferreira, J., Oliveira, H. G., and Rodrigues, R. (2019). Improving NLTK for processing Portuguese. In *8th Symposium on Languages, Applications and Technologies*, Coimbra, Portugal.

[Fonseca and Rosa 2013] Fonseca, E. R. and Rosa, J. L. G. (2013). Mac-morpho revisited: Towards robust part-of-speech tagging. In *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology*, Fortaleza, Brazil. SBC.

[Fortuna and Nunes 2018] Fortuna, P. and Nunes, S. (2018). A survey on automatic detection of hate speech in text. *ACM Computing Surveys (CSUR)*, 51(4):1–30.

[Goldberg 2017] Goldberg, Y. (2017). *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers.

[Jurafsky and Martin 2009] Jurafsky, D. and Martin, J. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall series in artificial intelligence. Pearson/Prentice Hall.

[Kalchbrenner et al. 2014] Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Baltimore, Maryland.

[Kaliyar et al. 2021] Kaliyar, R. K., Goswami, A., and Narang, P. (2021). Fakebert: Fake news detection in social media with a bert-based deep learning approach. *Multimedia Tools and Applications*, 80(8):11765–11788.

[Kim 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar. Association for Computational Linguistics.

[Kinoshita et al. 2007] Kinoshita, J., Salvador, L., Menezes, C., and Silva, W. (2007). Cogroo - an openoffice grammar checker. In *Proc. of the 17th International Conference on Intelligent Systems Design and Applications*, Rio de Janeiro, Brazil. IEEE.

[Le and Mikolov 2014] Le, Q. and Mikolov, T. (2014). Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1188–1196, Bejing, China. PMLR.

[LeCun 1989] LeCun, Y. (1989). Generalization and network design strategies. Technical report, University of Toronto.

[Leite et al. 2020] Leite, J. A., Silva, D., Bontcheva, K., and Scarton, C. (2020). Toxic language detection in social media for Brazilian Portuguese: New dataset and multilingual analysis. In *Proc. of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, Suzhou, China.

[Liu 2012] Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167.

[Liu 2015] Liu, B. (2015). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge University Press.

[Magalhães Jr et al. 2019] Magalhães Jr, G. V., Vieira, J. P. A., Santos, R. L. S., Barbosa, J. L. N., Santos Neto, P., and Moura, R. (2019). A study of the influence of textual features in learning medical prior authorization. In *Proc. of the 32nd International Symposium on Computer-Based Medical Systems*, Córdona, Spain. IEEE.

[Manning et al. 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[Meneses Silva et al. 2021] Meneses Silva, C. V., Silva Fontes, R., and Colaço Júnior, M. (2021). Intelligent fake news detection: a systematic mapping. *Journal of applied security research*, 16(2):168–189.

[Mikolov et al. 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *Proc. of the 1st International Conference on Learning Representations Workshop papers*, Scottsdale, AZ, USA.

[Mikolov et al. 2013b] Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *Proc. of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Atlanta, Georgia.

[Pedregosa et al. 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12.

[Poria et al. 2016] Poria, S., Cambria, E., and Gelbukh, A. (2016). Aspect extraction for opinion mining with a deep convolutional neural network. *Knowledge-Based Systems*, 108(C):42–49.

[Qi et al. 2020] Qi, P., Zhang, Y., Zhang, Y., Bolton, J., and Manning, C. D. (2020). Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Online.

[Rumelhart et al. 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[Santos et al. 2021] Santos, R. L. S., Sa, C. A., Sousa, R. F., Anchiêta, R. T., Rabelo, R. A. L., and Moura, R. S. (2021). Estimating importance from web reviews through textual description and metrics extraction. In *Natural Language Processing for Global and Local Business*. IGI Global.

[Sardinha 2000] Sardinha, T. B. (2000). Linguística de corpus: histórico e problemática. *DELTA: Documentação de Estudos em Lingüística Teórica e Aplicada*, 16:323–367.

[Sarkar 2019] Sarkar, D. (2019). *Text Analytics with Python. A Practitioner's Guide to Natural Language Processing*. APress.

[Soares and Moura 2015] Soares, H. A. and Moura, R. S. (2015). A methodology to guide writing software requirements specification document. In *Proceedings of the 2015 Latin American Computing Conference*, pages 1–11, Arequipa, Peru. IEEE.

[Sousa 2015] Sousa, R. F. (2015). Abordagem TOP(X) para inferir os comentários mais importantes sobre produtos e serviços. Master's thesis, Universidade Federal do Piauí.

[Vaswani et al. 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, Long Beach, USA.

Capítulo 2

Propriedade Intelectual e registro de programa de computador – Inovação e tecnologia na indústria moderna

Luana de Oliveira Lopes, Alcemir Rodrigues Santos

Abstract

The intellectual property (IP) refers to regulation acts that assure not only the authorship of human creations, but also it contributes to the creative/productive process inherent to the economic relations and the technological growth. Thus, the science and technology professionals should be aware of the regulatory acts and to the protection processes of their inventive activity. Such awareness contributes to them to be on the spot in the different job markets and it generates innovation assets to their company. Therefore, the economic exploitation of their creations and the promotion of the technological development contribute to the creative economy advance, which is essential to the industry 4.0.

Resumo

A propriedade intelectual (PI) refere-se à área do direito que garante não apenas a proteção de autoria das criações humanas, mas também contribui para o processo criativo/produtivo que permeia as relações econômicas e o avanço tecnológico. O profissional de ciência e tecnologia deve, portanto, estar atento às normas regulatórias e ao processo de proteção da sua atividade inventiva. Isso permite a este profissional um maior destaque no mercado de trabalho e gera capital de inovação às empresas. Nesse contexto, a exploração econômica das criações e a promoção do desenvolvimento tecnológico contribuem para o avanço da economia criativa que é essencial na indústria 4.0.

2.1. Introdução

A propriedade intelectual (PI) refere-se à área do direito que garante não apenas a proteção de autoria das criações humanas, mas também contribui para o processo criativo/produtivo que permeia as relações econômicas e o avanço tecnológico. O avanço tecnológico não

é estimulado somente pela introdução de novas técnicas, conceitos ou ferramentas, mas também pela valorização de estratégias de controle e qualidade que garantem a proteção da atividade inventiva e a transferência de tecnologia. As garantias de direito de propriedade intelectual são essenciais para manutenção e promoção da inovação de autores e terceiros envolvidos. Nesse contexto, abordaremos durante esse capítulo os conceitos, normas, regulamentos e desafios da propriedade intelectual, inclusive no que diz respeito aos registros de programas de computador e seu licenciamento.

O restante do capítulo está organizado da seguinte maneira. Primeiramente, apresentamos o contexto histórico em que surge a propriedade intelectual (Seção 2.2). Em seguida, apresentamos a propriedade intelectual e sua relevância para inovação e tecnologia (Seção 2.3). A Seção 2.4 apresenta a legislação e desafios regulatórios no registro de propriedade intelectual de forma geral. Já na Seção 2.5 trazemos o registro de programas de computador, o seu licenciamento, bem como o processo para pedidos de registro de software e detalhes sobre licenças de software livre e/ou de código aberto. Por fim, na Seção 2.6 dicutimos a indústria criativa e perspectivas futuras na indústria 4.0 para profissionais de tecnologia.

2.2. Contexto Histórico

A história tem mostrado a importância da proteção da propriedade intelectual e sua repercussão no desenvolvimento tecnológico, social e cultural das nações. Embora o início da atividade inventiva não possa ser identificado ao longo do tempo, muitas são as obras que foram notadamente marcos científicos e culturais protegidos pela legislação de propriedade intelectual, tais como as obras de Thomas Edison, Graham Bell, Benjamin Franklin, Leonardo Da Vinci, Santos Dumont, Isaac Newton (Figura 2.1).

Sabe-se que a primeira lei promulgada para a proteção de propriedade intelectual data do século XV [Jungmann 2010]. Em 1477, o governo da República de Veneza, local de intenso comércio e cultura, começou a emitir as primeiras cartas de patentes, para incentivar a produção e as contribuições aos Estados. A partir de então muitos foram os países que adotaram leis semelhantes. Somente por volta do século XVII, a Inglaterra legislou nessa área, onde promulgou a sua primeira lei de propriedade intelectual através do Estatuto dos Monopólios, seguida pelos Estados Unidos através do Patent Act já em 1790. No Brasil, por circunstância da transferência forçada da coroa de Portugal para o Brasil, o príncipe Regente assina a Carta Régia em 1808. Essa medida permitiu a abertura dos portos do Brasil às nações amigas e em 1809 foi criada a primeira legislação sobre patentes industriais no país [Kappeler 2005].

O desenvolvimento tecnológico nos séculos seguintes permitiu a elaboração de convenções em termos de propriedade industrial, com o objetivo de inibir a pirataria e incentivar a produção de invenções que viessem a contribuir aos Estados e governantes. Em 1883, foi estabelecida e assinada a Convenção de Paris que está em vigor até os dias atuais. O acordo, no entanto, passou por diversas modificações introduzidas no texto original por meio de seis revisões: Bruxelas (1900), Washington (1911), Haia (1925), Londres (1934), Lisboa (1958) e Estocolmo (1967). O Brasil, país signatário original, aderiu à Revisão de Estocolmo em 1992.

Além disso, após a Segunda Guerra Mundial, o surgimento do GATT (*General*

Figura 2.1. Grandes inventores da humanidade. a) Thomas Edison (inventor do fonógrafo); b) Santos Dumont (primeiro a decolar um objeto mais pesado que o ar); c) Benjamin Franklin (inventor da máquina eletrostática); d) Graham Bell (fundador das companhias telefônicas); e) Isaac Newton (construiu o primeiro telescópio refletor prático); f) Leonardo Da Vinci (pintor da "A Última Ceia" e "Mona Lisa").

Agreement on Tariffs and Trade) fez repercurtir ainda mais os temas de regulamentação de comércio e desenvolvimento econômico no mundo pós-guerra. Assim, em 1967 foi criada a Organização Mundial da Propriedade Intelectual (OMPI) pela Organização das Nações Unidas (ONU) [Jungmann 2010].

2.3. Propriedade Intelectual

Ao contrário do que se imagina, a propriedade intelectual não se restringe aos pedidos de depósito de patente. Para além desse registro, existem diversas formas de garantir a proteção das criações humanas e o seu potencial de exploração econômico e social. A OMPI refere-se à propriedade intelectual como:

> "(...) criações da mente, como invenções; obras literárias e artísticas; designs; e símbolos, nomes e imagens usados no comércio. A propriedade intelectual é protegida por lei, por exemplo, por patentes, direitos autorais e marcas registradas, que permitem às pessoas obter reconhecimento ou benefício financeiro com o que inventam ou criam."
>
> *[da Silva et al. 2021]*

Ao encontrar o equilíbrio certo entre os interesses dos inventores e o interesse público mais amplo, o sistema de PI visa promover um ambiente no qual a criatividade e a inovação possam florescer. A propriedade intelectual abrange, portanto, pelo três formas de proteção gerais que dispõem de particularidades de processo de registro, regulação e acesso e que são necessários à manutenção e promoção do conhecimento no

mundo. Incluem-se nesse repertório a **propriedade industrial**, com a proteção de patentes, marcas, desenho industrial, indicações geográficas, segredo industrial e repressão à concorrência desleal; os **direitos autorais** que incluem os programas de computador, obras literárias e artísticas; a **proteção *sui generis*** que inclui a proteção de cultivares, o conhecimento tradicional e a topografia de circuitos integrados. Esse sistema de proteção à propriedade intelectual visa estimular novas criações e incentivar o avanço tecnológico [Bagnato et al. 2016]. A Figura 2.2 apresenta um resumo das formas de propriedade intelectual.

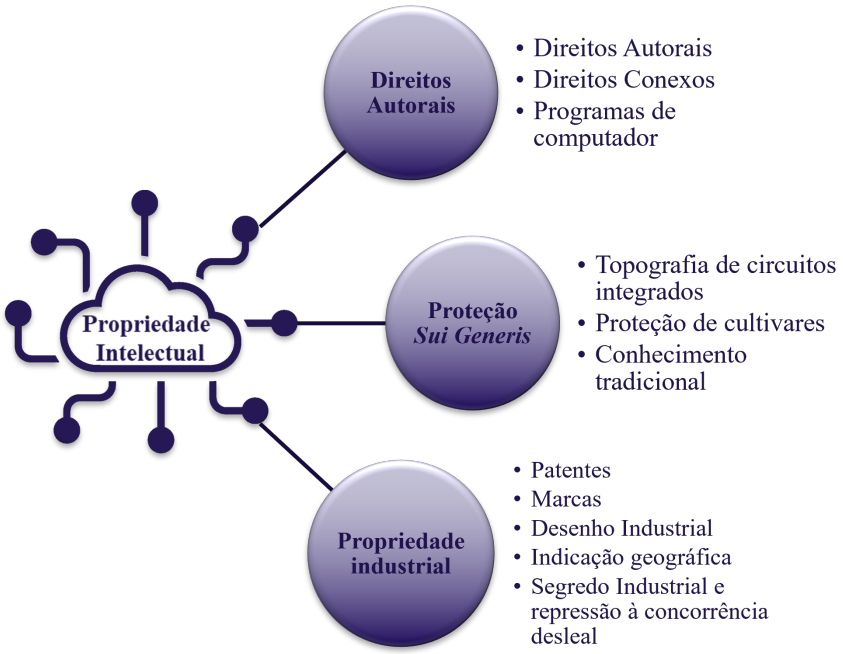


Figura 2.2. Resumo das formas de propriedade intelectual protegidas pela legislação brasileira.

Dentro de um contexto mercadológico, um mesmo produto pode conter diversos tipos de proteção. Por exemplo, um computador pode ter o *design* protegido pelo **desenho industrial**, ser exclusivo de uma determinada **marca**, um hardware que facilite alguma tecnologia protegido por **patente**, rodando um **software registrado**. Todos esses planos de proteção geram valor de mercado ao criador e as empresas envolvidas na transferência dessas tecnologias.

Assim, os direitos de propriedade asseguram direitos sobre bens de natureza material de forma clara ao seu titular e ainda garantem que os bens de natureza imaterial sejam usufruídos pela sociedade respeitando-se o prazo legal de exploração desses direitos. Isso gera vantagens competitivas e promove desenvolvimento e livre concorrência. A concessão de registros e direitos de propriedade são realizadas pelo Instituto Nacional de Propriedade Industrial (INPI) que também regulamenta o registro de marcas, desenhos industriais, indicações geográficas, programas de computador, topografias de circuitos integrados e contratos de tecnologia e de franquia [da Silva et al. 2021].

2.3.1. Patentes

As patentes são as mais conhecidas formas de proteção da propriedade intelectual que asseguram ao inventor o direito temporário de exploração de sua atividade inventiva. A exploração econômica desse patrimônio pode ser feita pelo próprio inventor ou transferida a terceiros por meio de pagamento de *royalties*. Para garantir o direito integral da titularidade é necessário a definição de parâmetros e regras de registro. Neste contexto, as patentes podem ser divididas em duas classes: as patentes de invenção e os modelos de utilidade. Para tanto, a Lei de Propriedade Industrial (LPI) - Lei nº 9.279, implementada em 1996 [Brasil 1996], estabelece que: "*Art. 8º É patenteável a invenção que atenda aos requisitos de novidade, atividade inventiva e aplicação industrial.*" Já os modelos de utilidade são objetos de uso prático ou parte deles que trazem melhorias com aplicação industrial a uma determinada patente ou a sua funcionalidade. Existe também um Certificado de Adição de Invenção, para proteger um aperfeiçoamento introduzido na matéria requerida pelo inventor em um pedido ou mesmo na patente já concedida.

Os critérios para pedidos de depósito de patente estão discriminados em lei e devem ser submetidos a apreciação pelo INPI. As patentes de invenção ou modelos de utilidade são considerados novos quando não encontradas no *estado da técnica* ou *estado da arte*, ou seja, que não tenham sido patenteadas ou divulgadas anteriormente seja por via oral ou escrita. Sob esse ponto de vista, deve-se observar também a atividade inventiva e aplicação industrial para redação do pedido de patente. Considera-se invenção ou produto ou processo contendo atividade inventiva quando, para um técnico no assunto, aquela invenção não decorra de maneira óbvia e evidente do estado da técnica. E por último, uma invenção apresenta aplicação industrial quando possa ser utilizado ou produzido por qualquer tipo de indústria [da Silva et al. 2021].

Vale ressaltar, que muitas descobertas não podem ser patenteadas (Figura 2.3) e, não se incluem nos registros de patentes, mas podem ser protegidos por legislações específicas que asseguram ao inventor os direitos de titularidades para valorização pessoal, cultural e financeira da qual decorrem suas invenções.

Figura 2.3. Resumo de bens materiais que não podem ser patenteados.

Outra qualidade imprescindível nos pedidos de depósito de patente são as redações desses documentos de apresentação. As patentes devem conter informações mínimas de conceitualização e contextualização do problema a ser solucionado. A invenção ou novidade deve se introduz em um relatório descritivo do processo de produção da invenção. Segundo a LPI [Brasil 1996], *O relatório deverá descrever clara e suficientemente o objeto, de modo a possibilitar sua realização por técnico no assunto e indicar, quando for o caso, a melhor forma de execução* [Brasil 1996, Art. 24]. Além disso, a redação da patente deve trazer uma vasta pesquisa a respeito do estado da arte e suas as consequências da introdução dessa invenção para a melhorias dos processos ou produtos já conhecidos.

Embora as demais partes sejam importantes, nada é mais característico da redação de patentes como as **reivindicações**. Essa sessão da redação deve conter as partes a serem protegidas com a maior grau de especificação e detalhes possível. A extensão da proteção conferida pela patente é determinada pelo conteúdo das reivindicações, interpretado com base no relatório descritivo e nos desenhos, ou seja, as reivindicações definem e delimitam os direitos do autor do pedido [Brasil 1996, Art. 41]. Por esse motivo, são essenciais para garantir ao titular da patente a abrangência da utilização do objeto da patente.

Após a redação do pedido de patente, este deve ser depositado no INPI juntamente com a documentação necessária para análise jurídica a respeito dos requisitos legais. Assim, será realizada uma análise do estado da técnica e demais informações necessárias à concessão. Inicialmente é feita uma análise formal no período de 30 dias contados a partir da data de depósito, principalmente no que diz respeito aos requisitos e formulários básicos. Não sendo encontradas pendências, o pedido é aceito e entra na fase de sigilo de 18 meses, dentro dos quais o inventor pode retirar o pedido sem divulgação da invenção. Após esse período o pedido é publicado na revista do INPI e o inventor tem até 36 meses para realizar o pedido de exame de mérito. No exame de mérito, serão verificados a novidade e atividade inventiva e aplicação industrial do pedido e as exigências formuladas devem ser respondidads em até 60 dias. Quando houver o indeferimento, ou seja, parecer negativo à proteção, o depositante poderá apresentar recurso em até noventa 90 dias. Em caso de deferimento, o INPI emite uma outra taxa que deve ser paga em até 60 dias e só então há a concessão da patente. Vale ressaltar também que o inventor deve atualizar a taxa de anuidade garantindo a proteção pelo tempo de até 20 anos no caso de invenção e 15 anos, no caso de modelo de utilidade [da Silva et al. 2021].

Qualquer pessoa física ou jurídica pode solicitar um pedido de depósito de patentes junto ao INPI. Esse pedido pode ser realizado via eletrônica ou por escrito, de acordo com os formulários disponibilizados pelo escritório. No site do INPI, todas as informações necessárias a este pedido estão disponíveis e, em linhas gerais, a solicitação do pedido de patentes ocorre em por meio do cadastramento, seguido da geração e pagamento da GRU e em seguida, peticionamento e processamento do exame.

As patentes são instrumentos de negócios e valorização da criatividade, não são meros títulos de propriedade. O titular tem o direito de impedir terceiros de produzir, usar, colocar à venda, vender ou importar, sem o seu consentimento, o produto objeto de patente ou processo ou produto obtido diretamente por processo patenteado. Por esse motivo, após três anos da concessão, se não houver comercialização ou licenciamento ou o titular negar-se a negociar a patente, ou ainda, houver abuso de poder econômico, poderá

haver a licença compulsória a terceiros sem exclusividade, sendo o titular remunerado segundo decisão arbitrada pelo INPI [Peixoto and Buainain 2021].

2.3.2. Marcas

Segundo o INPI [da Silva et al. 2021], as marcas são sinais distintivos que visam identificar a origem e distinguir produtos ou serviços que são idênticos de outros, semelhantes ou afins de origem diversa. Atualmente, as marcas são ferramentas de valorização da atividade comercial. Nesse contexto, as marcas representam parte da estratégia de ligação da marca ao consumidor gerando valorização da empresa e guiando a atividade mercadológica. As marcas são mais que um conjunto de símbolos e/ou palavras, pelo contrário são parte da experiência do consumidor.

Em relação a sua formatação gráfica, as marcas podem ser classificadas como nominativa, figurativa, mista e tridimensional de acordo com a Figura 2.4.

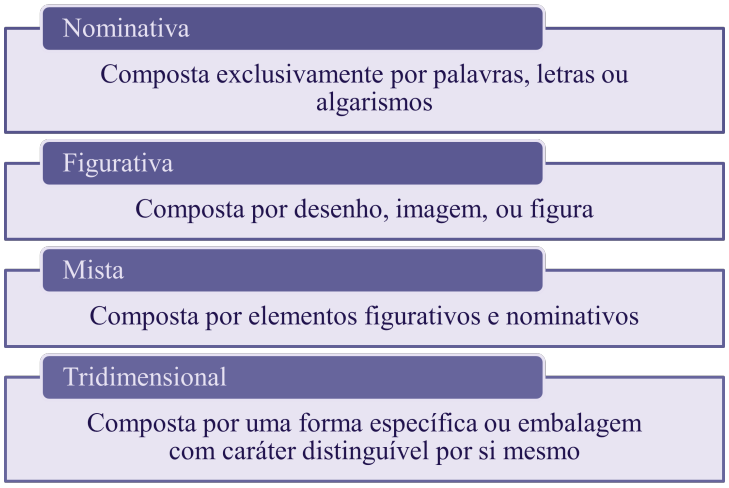


Figura 2.4. Classificação dos tipos de marca com relação à sua formatação gráfica [Taddei 2010].

De acordo com a sua natureza, as marcas podem ser classificadas como de produto ou serviço, de certificação ou coletivas de acordo com os critérios a da Figura 2.5.

Como proteger uma marca no Brasil? Assim como as patentes, qualquer pessoa física ou jurídica, de direito público ou privado pode solicitar registro de marcas. No entanto, é necessário observar a natureza da marca e do requerente ao submeter o pedido de registro. Segundo o art. 128 da LPI [Brasil 1996]:

> “ As pessoas de direito privado só podem requerer registro de marca relativo à atividade que exerçam efetiva e licitamente, de modo direto ou através de empresas que controlem direta ou indiretamente, declarando, no próprio requerimento, esta condição, sob as penas da lei.
>
> *[Brasil 1996]* ”

Além disso, somente pode solicitar uma marca de certificação o requerente (pessoa jurídica) que não possuir interesse financeiro ou jurídico na aquisição da certificação.

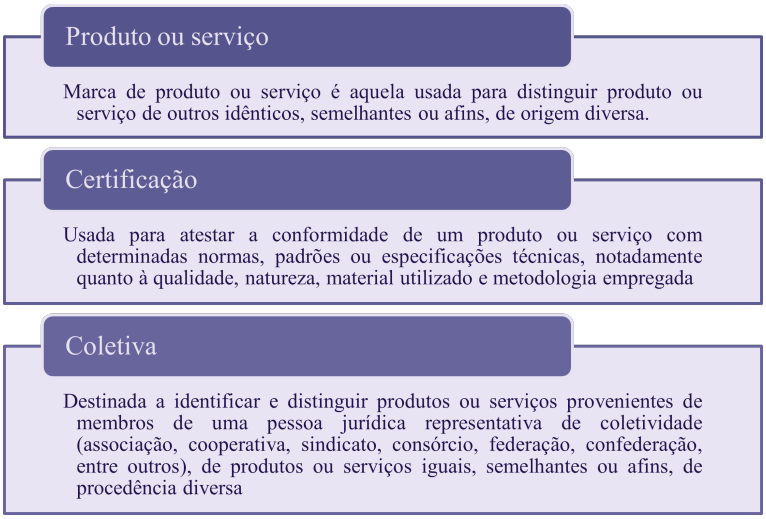


Figura 2.5. Classificação dos tipos de marca com relação à sua natureza [Taddei 2010].

A legislação inclui outros critérios de habilitação, por isso é importante um acompanhamento jurídico ou de especialistas para a requisição.

No que diz respeito ao símbolo distintivo, alguns critérios também devem ser observados ao requerimento desta proteção, para garantir um uso adequado e evitar conflitos de interesse. Abaixo, estão elencados alguns desses critérios de exclusão, segundo INPI [INPI 2013]:

Termos genéricos: Não é possível proteger uma marca que utilize termos gerais que representem um produto ou serviço em si. Por exemplo, a requisição da proteção da marca "ESPELHO" provavelmente seria negada devido ao termo genérico que compõe o objeto do produto.

Termos descritivos: São as palavras que descrevem o produto ou serviço solicitado, geralmente adjetivos que também são genéricos e tendenciosos.

Marcas falaciosas: Símbolos que tentem confundir o consumidor por induzir um caráter que não está apresentado no produto/serviço.

Marcas consideradas contrárias à ordem pública ou à moral: Há que se garantir a divulgação e veiculação do símbolo do requerente o que não condiz com a utilização de termos e símbolos que vão de encontro à ordem pública e moral.

Bandeiras, escudos de armas, carimbos oficiais e emblemas de Estados e de organizações internacionais que tenham sido comunicados à Organização Mundial da Propriedade Intelectual: por óbvio, também são geralmente excluídos do registro.

Embora uma marca possa ser protegida pelo seu uso no mercado, ou seja, pela associação intuitiva de um produto ou serviço à marca dada vasta divulgação no mercado, recomenda-se fortemente o seu registro. Essa certificação garante que não haja conflitos e/ou utilização indevida na comercialização de produtos ou serviços semelhantes.

No Brasil, esse registro também é concedido pelo INPI e não deve se confundido

com o nome empresarial. O registro de marcas diz respeito à um símbolo ou nome que representa o produto/serviço ou parte de dele, que como observado anteriormente pode ser representado inclusive por um formato 3D. Por outro lado, o nome empresarial diz respeito ao nome utilizado em transações comerciais e geralmente trazem o enquadramento jurídico a que se referem ao final como por exemplo, Ltda. ou S.A.

Para realizar o pedido de registro é importante, além de respeitar os critérios legais já apresentados, realizar uma busca de anterioridade que pode ser feita inclusive no site do INPI[1].

O requerente deve então fazer seu cadastro no site do INPI e gerar a guia de recolhimento da união - GRU referente a taxa de registro de marcas e em seguida apresentar um formulário online ou por escrito junto ao INPI. Além desses documentos, devem ser apresentadas as imagens com a representação gráfica da marca, bem como a descrição detalhada do produto ou serviço referente à marca solicitada.

Assim como nas patentes, o instituto realizará um **exame formal** para verificar os requisitos legais e a possibilidade de registro. No exame formal, é verificado se há discrepâncias entre os dados informados pelo requerente do pedido no que diz respeito à marca e sua apresentação, prioridade, procurador, atividade declarada, bem como demais documentos anexados pelo peticionário. Se houver alguma inconformidade, o INPI publica uma notificação que deve ser respondida em até 5 dias úteis. Após o exame formal, caso não haja nenhuma inconformidade, o pedido é publicado na **Revista da propriedade Industrial** (RPI) pelo prazo de até 60 dias para investigação de oposição. Após esse período, após a fase de publicação e eventuais oposições de terceiros, inicia-se a fase de **exame de mérito**. Nessa etapa serão verificados se os requisitos de registrabilidade de uma marca foram devidamente atendidos. Em caso positivo, publica-se o deferimento da mesma para que o titular proceda, dentro de 60 (sessenta) dias, ao pagamento das taxas finais. Com os devidos pagamentos realizados, é publicada a concessão do registro e emitido o respectivo Certificado, válido pelo período de dez anos. Esse registro pode ser prorrogado indefinidamente, devendo-se solicitar a renovação e realizar-se os pagamentos das taxas referentes aos decênios seguintes [Peixoto and Buainain 2021].

É importante ressaltar que, no Brasil, cada pedido está limitado a uma única classe. Portanto, se a marca for requerida para produtos ou serviços de classes diferentes, será necessário apresentar um pedido para cada classe.

2.3.3. Proteção por desenho industrial

Outro tipo de proteção de propriedade industrial que pode ser realizado via INPI é o desenho industrial. Esse tipo de PI trata do desenho associado à forma plástica ornamental de um objeto ou ao conjunto ornamental de linhas e cores que possa ser aplicado a um produto. As formas gráficas que incluem características tridimensionais, ou bidimensionais, como padrões, linhas ou cores e que proporcionam resultado visual novo e original na sua configuração externa são passíveis de proteção por desenho industrial. No Brasil, quem concede o registro é o INPI, e sua validade é de até 25 anos. Como regra geral, para ser registrável, o desenho precisa atender aos requisitos de: novidade, originalidade

[1]Disponível em: `https://busca.inpi.gov.br/pePI/jsp/marcas/Pesquisa_num_processo.jsp`

e utilização ou aplicação industrial. Para requerer um depósito do pedido de registro do desenho industrial, é necessário:

- Requerimento e formulário de cadastro;
- Relatório descritivo, se for o caso;
- Reivindicações, se for o caso;
- Desenhos ou fotografias;
- Campo de aplicação do objeto;
- Comprovante do pagamento de taxas.

2.3.4. Proteção por indicação geográfica

Ainda sobre propriedade industrial, observou-se a importância econômica associada à valorização de produtos advindos de regiões específicas com qualidades e/ou reputação únicas relacionadas à sua forma de extração, produção ou fabricação. As indicações geográficas (IG) são classificadas em: Denominação de origem (DO) e Indicação de procedência (IP). A IP é o nome geográfico de país, cidade, região ou localidade de seu território, que se tenha tornado conhecido como centro de extração, produção ou fabricação de determinado produto ou de prestação de determinado serviço. Já a DO é o nome geográfico de país, cidade, região ou localidade de seu território, que designe produto ou serviço cujas qualidades ou características se devam exclusiva ou essencialmente ao meio geográfico, incluídos fatores naturais e humanos.

O depósito do pedido de registro para uma indicação geográfica, nas condições estabelecidas pela LPI (1996), precisa conter:

- Dados do requerente;
- Espécie de indicação geográfica pretendida;
- Nome da área geográfica;
- Natureza do objeto da proteção (produto ou serviço);
- Delimitação da área geográfica;
- Objeto do produto ou serviço produzido na área delimitada;
- Comprovante do pagamento de taxas.

Além dos documentos e informações acima referidos, o pedido deverá apresentar informações e provas específicas, de acordo com a espécie de indicação geográfica pleiteada.

2.3.5. Segredo industrial e proteção contra a concorrência desleal

O segredo industrial corresponde a informações de produtos e serviços que contenham valor comercial e sejam secretas para a empresa podem ser protegidas pela LPI. Essas informações devem, além de secretas, serem pouco acessíveis a pessoas de círculos que normalmente lidam com o tipo de informação em questão, e devem ter sido objeto de precauções razoáveis, nas circunstâncias, pela pessoa legalmente em controle da informação, para mantê-la secreta.

Divulgar informações confidenciais ou explorar ou utilizar, sem autorização ou por meios ilícitos (segredo de negócio) empregáveis na indústria, comércio ou prestação de serviços é o crime, previsto na LPI (1996), chamado concorrência desleal. Também constitui concorrência desleal o acesso a informações mediante relação contratual ou empregatícia, mesmo após o término do contrato. É importante ressaltar que não são considerados crimes pela LPI a divulgação, exploração ou utilização dos conhecimentos e informações ou dados que sejam públicos ou evidentes para um técnico no assunto.

2.4. Direito e Inovação

As garantias legislativas nacionais e internacionais são essenciais para promover a valorização da propriedade intelectual, seja sob o aspecto comercial, social ou cultural. O desenvolvimento tecnológico está intimamente relacionado à boa gestão e implementação dessas normas e tendências na atividade produtiva e inventiva. Como foi possível observar nas sessões anteriores, os direitos de propriedade intelectual são bem definidos e englobam uma série de peculiaridades regulamentadas por leis e decretos, inclusive no âmbito da discriminação de deveres dos órgãos competentes. Todos esses mecanismos jurídicos visam esclarecer e legitimar as invenções e obras produzidas pelos titulares e garantir o usufruto dos seus bens pelo tempo que melhor convém à sociedade. Nessa sessão serão apresentados os principais mecanismos jurídicos e acordos internacionais que regem o direito de propriedade intelectual e incentivam a inovação no Brasil e no mundo [Peixoto and Buainain 2021].

2.4.1. Legislação brasileira em PI e acordos internacionais

Embora o cumprimento e regularização de PI seja essencial no desenvolvimento de qualquer país, as estratégias de divulgação deste tema ainda são pouco representativas em países emergentes, como o Brasil. O acesso à informação técnico/tecnológica guia políticas públicas de inovação, que repercutem sobre áreas como comércio exterior, segurança, biodiversidade, produção de bens e comunicação. Os acordos internacionais são notadamente complexos e heterogêneos quanto a sua elaboração e finalidades, uma vez que possuem particularidades regionais e sociais principalmente no que diz respeito à biodiversidade e ao comércio [de Mello e Souza et al. 2014]. Por esse motivo, a elaboração e colaboração entre os países é essencial para a manutenção das boas práticas comerciais e produtivas mundialmente.

Ao mesmo tempo em que esses acordos são essenciais no sentido organizacional e são consistentes com a divulgação de produtos, o acesso amplo à informação e o combate à concorrência desleal e a pirataria, estes possuem vieses de limitação. A autonomia para gerir a extensão e especificação das obrigações relativas ao escopo, ao objeto e à duração

da proteção de PI geram conflitos entre os países signatários tendo em vista suas particularidades geográficas e comerciais. Esses desafios tendem a ser explorados em convenções e acordos que se tornam cada vez mais essenciais na manutenção da necessidade de respeitar a diversidade dos países envolvidos.

Apesar de a propriedade intelectual ser reconhecida a séculos, os direitos concedidos aos titulares eram restritos às fronteiras de seus respectivos países. Com a Convenção da União de Paris CUP (1883) e a Convenção de Berna (1886) foi possível mudar esse cenário. Estes acordos estabeleceram medidas relativas ao usufruto dos privilégios de invenção concedidos aos inventores em cada um dos Estados signatários, bem como no seu estado de origem. A CUP também prevê o direito de **prioridade unionista** no caso de patentes (e modelos de utilidade onde existem), marcas e desenhos industriais. Assim, foi possível estabelecer o direito de prioridade de depósito independente do país de origem, dentro dos países contratantes. O Brasil ratificou este acordo por meio da Lei 376/1896, que aprova os quatro protocolos formulados na conferência de Madrid em abril de 1890.

Outro evento de destaque envolvendo direito internacional e PI é o Acordo Geral sobre Tarifas e Comércio (em inglês, GATT). Suas rodadas foram de extrema importância para o diálogo e consolidação das normas internacionais. A primeira rodada foi estabelecida em 1947, durante a Conferência das Nações Unidas sobre Comércio e Emprego, em Havana. Nesse acordo, houve incentivo para obtenção de vantagens mútuas por meio de reduções tarifárias e de barreiras comerciais, bem como eliminação de preferências entre os países signatários, entre eles o Brasil.

A seguir, somente com a Convenção de Estocolmo (1967), a CUP é promulgada e é responsável pelo diálogo internacional que instituiu a Organização Mundial de Propriedade Intelectual (OMPI). A OMPI foi um marco na área de PI por atuar na atualização e proposição de padrões internacionais de proteção às criações intelectuais em âmbito mundial. Por meio da OMPI foram criados: o Tratado de Cooperação em Matéria de Patentes (PCT), de 1970, cuja modificação mais recente foi em 2001; Apoio à Convenção para a Proteção de Novas Variedades de Vegetais, que em 1961 deu origem à União Internacional para a Proteção das Obtenções Vegetais (UPOV), e o Sistema de Registro Internacional de Marcas, regido pelo Acordo de Madri relativo ao Registro Internacional de Marcas, de 1981, e pelo Protocolo referente ao Acordo de Madri (Protocolo de Madri), de 1989, que começou a ser aplicado em 1996, e reúne 104 países [Peixoto and Buainain 2021].

Somente ao final da Rodada Uruguai do GATT em 1994 é assinado o Acordo sobre os Aspectos dos Direitos de Propriedade Intelectual Relacionados com o Comércio (em inglês, TRIPS), essencial na regulamentação de transferência de tecnologia em PI no exterior. Esse acordo também gerou a criação da Organização Mundial do Comércio. É importante ressaltar que a legislação brasileira respeita todos esses acordos internacionais sobretudo o TRIPS, que entrou em vigor em janeiro de 1995 aqui no Brasil.

O arcabouço jurídico brasileiro é bastante amplo no que se refere a propriedade intelectual (Tabela 2.1). Consistente com essa informação, é essencial compreender os direitos em propriedade intelectual como a execução do direito de propriedade em si, ou seja, por meio deles é possível usar, fruir, dispor, explorar e ceder o exercício do domínio sobre algo. Destaca-se em primeiro lugar a criação do INPI (Instituto Nacional da Propriedade Industrial) pela Lei 5.648/70. Esse órgão representa uma autarquia federal,

vinculada ao Ministério do Desenvolvimento, Indústria e Comércio Exterior e executa as normas que regulam a propriedade industrial em âmbito nacional. Além disso, seguem os direitos de propriedade industrial que abrangem patentes, marcas, desenho e modelo industrial, indicações geográficas, segredo industrial e repressão à concorrência de acordo com a Lei 9.279/96; os direitos autorais e conexos, compreendendo as obras literárias, artísticas e científicas, interpretações dos artistas e intérpretes e execuções dos artistas e executantes, os fonogramas e as emissões de rádio difusão, sendo protegidos pela Lei 9.610/98; a proteção aos programas de computadores, regulamentada pela Lei 9.609/98; as proteções sui generis, como cultivares, topografias de circuitos fechados e conhecimento tradicional sendo protegidos respectivamente pela Lei 9.456/97, Lei 11484/07 e Medida Provisória 2.186-16/01.

Tabela 2.1. Legislação sobre propriedade intelectual no Brasil [dos Deputados 2010].

Dispositivo	**Ementa**
Decreto-Lei nº 2.848/1940	Código Penal.
Decreto-Lei nº 3.689/1941	Código de processo penal
Lei nº 9.279/1996	Regula direitos e obrigações relativos à propriedade industrial.
Lei nº 9.609/1998	Dispõe sobre a proteção da propriedade intelectual de programa de computador, sua comercialização no País, e dá outras providências
Lei nº 9.610/1998	Altera, atualiza e consolida a legislação sobre direitos autorais e dá outras providências
Lei nº 10.603/2002	Dispõe sobre a proteção de informação não divulgada submetida para aprovação da comercialização de produtos e dá outras providências
Lei nº 10.973/2004	Dispõe sobre incentivos à inovação e à pesquisa científica e tecnológica no ambiente produtivo e dá outras providências
Decreto nº 2.553/1998	Regulamenta o art. 75 e os arts. 88 a 93 da Lei nº 9.279, de 14 de maio de 1996, que regula direitos e obrigações relativos à propriedade industrial
Decreto nº 2.556/1998	Regulamenta o registro previsto no art. 3º da Lei nº 9.609/1998, que dispõe sobre a proteção da propriedade intelectual de programa de computador, sua comercialização no País, e dá outras providências
Decreto nº 3.201/1999	Dispõe sobre a concessão, de ofício, de licença compulsória nos casos de emergência nacional e de interesse público de que trata o art. 71 da Lei nº 9.279/1996
Decreto nº 9.931/2019	Institui o Grupo Interministerial de Propriedade Intelectual.
Decreto nº 4.533/2002	Regulamenta o art. 113 da Lei nº 9.610, de 19 de fevereiro de 1998, no que se refere a fonogramas, e dá outras providências

Decreto nº 5.244/2004	Dispõe sobre a composição e funcionamento do Conselho Nacional de Combate à Pirataria e Delitos contra a Propriedade Intelectual, e dá outras providências
Decreto nº 5.563/2005	Regulamenta a Lei nº 10.973/2004, que dispõe sobre incentivos à inovação e à pesquisa científica e tecnológica no ambiente produtivo, e dá outras providências
Lei nº 9.456/1997	Institui a Lei de Proteção de Cultivares e dá outras providências
Medida Provisória nº 2.186-16/2001	Regulamenta o inciso II do § 1º e o § 4º do art. 225 da Constituição, e os arts. 1º , 8º , alínea "j", 10, alínea "c", 15 e 16, alíneas 3 e 4 da Convenção sobre Diversidade Biológica, dispõe sobre o acesso ao patrimônio genético, a proteção e o acesso ao conhecimento tradicional associado, a repartição de benefícios e o acesso à tecnologia e a transferência de tecnologia para sua conservação e utilização, e dá outras providências
Lei nº 5.648/1970.	Regulamento, Cria o Instituto Nacional da Propriedade Industrial e dá outras providências

2.4.2. Direitos autorais

O direito autoral é a parte dos direitos em PI que protege legalmente a relação entre o criador e a sua obra, desde que a mesma seja de caráter estético, conduzindo a proteção de um bem imaterial. A atividade criativa protegida pelos direitos autorais visa colaborar não apenas ao desfrute no âmbito pessoal do autor, mas também, em uma dimensão mais abrangente, seu conjunto forma a herança cultural de um povo. Pode ser dividido quanto a sua natureza: em direitos morais do autor, que dizem respeito ao reconhecimento de criação e a paternidade da obra; em um segundo aspecto, em direitos patrimoniais que dizem respeito à exclusividade no uso e exploração de sua obra, bem como no direito de modificar, adaptar e distribui-la; e, por fim, a última diz respeito aos direitos conexos dos executantes e intérpretes.

Embora o registro de direitos autorais sejam facultativos no Brasil, eles são de extrema importância para o desenvolvimento e geração de renda no mundo. Levando em consideração a produção intelectual nos Estados Unidos, a indústria do direito autoral contribuiu com uma taxa de 22.74% para o crescimento real alçado pela economia em 2006 e 2007, podendo aumentar essa taxa se forem incluídos recursos de toda a cadeia produtiva até quase o dobro desse valor. No Brasil, o Direito Autoral em nosso país é tutelado pela Lei no 9.610/98. Apesar dos vieses de proteção internacional com vistas na colaboração de acordos, essa indústria tende a atender uma expectativa real de crescimento nos próximos anos [Jungmann 2010].

Segundo essa legislação, as obras devem possuir o requisito fundamental exigido de originalidade da obra criada, para que a mesma seja protegida pelo ordenamento jurídico. É o aspecto pessoal mais importante no direito do autor, tendo em vista tratar-se de direito personalíssimo, sendo, portanto, irrenunciáveis e inalienáveis (Art. 27 da Lei no 9.610/98), além de imprescritíveis e impenhoráveis. Os direitos de autor abrangem:

- Os textos de obras literárias, artísticas ou cientificas;
- As obras coreográficas e pantomímicas;
- As composições musicais;
- As obras fotográficas e as audiovisuais, inclusive as cinematográficas;
- As obras de desenho, pintura, gravura, escultura, litografia e arte cinética;
- As ilustrações, cartas geográficas e outras obras da mesma natureza;
- Os projetos, esboços e obras plásticas concernentes a geografia, engenharia, topografia,
- arquitetura, paisagismo, cenografia e ciência;
- As adaptações, traduções e outras transformações de obras originais, apresentadas
- como criação intelectual nova;
- As coletâneas ou compilações, antologias, enciclopédias, dicionários, bases de dados e
- e outras obras que se constituam uma nova criação intelectual;
- Os programas de computador.

Embora os direitos autorais sejam facultativos no Brasil, a proteção e registro é uma garantia contra a pirataria, reprodução e utilização indevida das obras. Atualmente, apesar da facilidade de divulgação e acesso de produtos por meio da internet, esse uso pode se tornar descontrolado e prejudicial, podendo gerar perdas financeiras significativas para os autores. Desse modo, a proteção por meio do registro de direitos autorais é um mecanismo de valorização e de garantia de qualidade, não apenas para os autores mas também àqueles que venham a explorar comercialmente essas obras.

2.5. Registro e Licenciamento de Software

Esta seção trata especificamente da propriedade intelectual relacionada à software, ou como é tratada na legislação vigente, programas de computador. Abaixo, apresenta-se a definição do termo, de acordo com o que consta na Lei.

Definição (Lei nº 9.609/1996 [Brasil 1998b])**.** *programa de computador (software) é a expressão de um conjunto organizado de instruções em linguagem natural ou codificada, contida em suporte físico de qualquer natureza, de emprego necessário em máquinas automáticas de tratamento da informação, dispositivos, instrumentos ou equipamentos periféricos, baseados em técnica digital ou análoga, para fazê-los funcionar de modo e para fins determinados. Esta definição também inclui os aplicativos desenvolvidos para smartphone.*

No quesito propriedade intelectual em software é possível tratarmos de dois conceitos importantes: o *registro* e o *licenciamento* de software. Mas, o que é o registro de software? E o que é o licenciamento de software? O *registro de programa de computador (RPC)* é a criação de prova de autoria do mesmo. Já o *licenciamento de software* é o meio pelo qual o gerador do programa de computador estabelece condições para o uso do mesmo e eventualmente sua redistribuição, modificação ou mesmo o sub-licenciamento por parte de terceiros. Ambos artefatos estão relacionados ao direito autoral de obras escritas na forma de programas de computador. No entanto, vale ressaltar que apesar das licenças de software serem acordos entre as partes geradora e consumidora, elas não são leis e portanto não estão acima dela.

Embora o registro de programa de computador não seja obrigatório, ele é fundamental para comprovar a autoria de seu desenvolvimento perante o Poder Judiciário. O regime de proteção à propriedade intelectual de programa de computador é o conferido às obras literárias pela legislação de direitos autorais e conexos vigentes no País. A Lei nº 9.609/1996 garante proteção legal de 50 anos ao titular do registro. A Tabela 2.2 apresenta a legislação atual que regula a proteção dos programas de computador. Enquanto a Lei nº 9.609/1996 especifica o que é considerado como programa de computador para a legislação nacional, ela também estabelece a proteção da propriedade intelectual destes através do seu registro e as formas de comercialização de software em território nacional. Por outro lado, o Decreto nº 2.556/1998 define o *Instituto Nacional da Propriedade Intelectual (INPI)* como o responsável pelo registro de programas de computador, passando ao mesmo o poder de emitir tanto normas adicionais para o regular o processo de registro, quanto o certificado de registro de programa de computador. Portanto, inclui-se o registro de software ao lado da proteção de patentes, marcas, desenho industrial, indicações geográficas e outras proteções discutidas neste capítulo ao conjunto de responsabilidades do INPI. Por fim, o próprio INPI emitiu em 2019 a Instrução Normativa nº 099 com o intuito de disciplinar o processo de registro de programas de computador.

Antigamente, o processo de RPC envolvia muita burocracia, incluindo a impressão de todo o código-fonte do programa e envio ao INPI. Atualmente, o processo é feito todo digitalmente através do sistema eletrônico do INPI, o *e-Software*, e o pedido pode ser feito pelo titular do direito ou por seu procurador. O INPI disponibiliza um manual para a submissão do pedido de registro [Alvares et al. 2019]. Em resumo, para realizar o depósito de RPC, se faz necessário a transformação dos trechos do programa de computador e de outros dados que considerar suficientes e relevantes para identificá-lo em resumo digital *hash* e o titular do direito é responsável pela guarda do objeto resumido. A Figura 2.6 apresenta um exemplo de certificado de registro de software emitido após a conclusão do RPC.

O direito autoral no Brasil define que a utilização da obra depende de autorização prévia e expressa do autor da obra originária. Em outras palavras, o uso de programa de computador no País é objeto de contrato de licença [Brasil 1998b, Art. 9º]. Para além do uso, também é possível criar uma obra nova a partir de uma outra original mediante prévia autorização de derivação de obra original. Neste caso, a obra derivada constitui a criação intelectual nova resultante da transformação da obra originária. Programas de computador derivados devem mencionar este fato explicitamente e detalhar, por exemplo, licenças originais do software modificado e/ou artefato legal que garante o permissionamento para

Tabela 2.2. Legislação vigente para registro de software.

Dispositivo	Ementa
Lei nº 9.609/1996	Dispõe sobre a proteção da propriedade intelectual de programa de computador, sua comercialização no País, e dá outras providências [Brasil 1998b].
Decreto nº 2.556/1998	Regulamenta o registro previsto no Art. 3º da Lei nº 9.609, de 19 de fevereiro de 1998, que dispõe sobre a proteção da propriedade intelectual de programa de computador, sua comercialização no País, e dá outras providências [Brasil 1998a].
Instrução Normativa nº 099/2019*	Disciplina o processo de registro eletrônico de programas de computador.

*Disponível em: `https://bit.ly/3xYfbxL` *(Acesso em 20 de Agosto de 2021.)*

a modificação. O autor do software derivado deve manter em sua posse os contratos de licença e/ou documento de autorização de modificação para sua segurança jurídica [Alvares et al. 2019].

Existem inúmeras licenças de software disponíveis. Especialmente, para licenciamento de software de código-aberto e sofware livre. A *Fundação Linux* mantém um projeto de padronização de enpacotamento de software (SPDX[2]). Para o caso de de licenciamento de software proprietário, sejam licença para uso temporário (*e.g.*, contrato de uso temporário, comum no caso de software de escritório) seja licença para uso perpétuo (*e.g.*, contrato de uso permanente de uso, como no caso de aplicativos para smartphones), é recomendável consultar um(a) profissional especialista na redação de contatos. Para efeitos didáticos, apresentamos algumas delas de forma resumida neste capítulo. A Tabela 2.3 apresenta oito diferentes versões de licenças de software.

De maneira complementar, a Tabela 2.4 apresenta um comparativo das permissões de cada uma destas licenças. A Tabela considera oito atributos (*i.e.*, associação, distribuição, modificação, uso comercial e sub-licenciamento, inclusão de *copyright*, liberdade de *royalties* e comunicação de mudanças obrigatória) para construir um um guia resumido dos termos e as condições de cada uma destas licenças. Os atributos considerados na Tabela 2.4 representam as seguintes permissões e limitações:

Associação: inclusão de referência no código utilizado juntamente com o código licenciado com uma licença diferente (*e.g.*, quando o código for disponibilizado como uma biblioteca);

[2]A página da SPDX está disponível em lingua inglesa no endereço: `https://spdx.org/` *(Acesso em 25 de Agosto de 2021.).*

REPÚBLICA FEDERATIVA DO BRASIL
MINISTÉRIO DA ECONOMIA
INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
DIRETORIA DE PATENTES, PROGRAMAS DE COMPUTADOR E TOPOGRAFIAS DE CIRCUITOS INTEGRADOS

Certificado de Registro de Programa de Computador

Processo Nº: BR5[redacted]7-5

O Instituto Nacional da Propriedade Industrial expede o presente certificado de registro de programa de computador, válido por 50 anos a partir de 1º de janeiro subsequente à data de 23/07/2020, em conformidade com o §2º, art. 2º da Lei 9.609, de 19 de Fevereiro de 1998.

Título: [redacted]

Data de publicação: 23/07/2020

Data de criação: 20/06/2020

Titular(es): [redacted]

Autor(es) [redacted]

Linguagem: HTML; JAVA SCRIPT; SQL; XML; CSS; NODEJS; OUTROS

Campo de aplicação: ED-04; ED-06

Tipo de programa: AP-01

Algoritmo hash: OUTROS

Resumo digital hash: AD18687584[redacted]0A865B

Expedido em: 28/07/2020

Aprovado por:
[redacted]
Chefe da DIPTO - Portaria/INPI/DIRPA Nº 09, de 01 de julho de 2019

Figura 2.6. Exemplo de certificado de retistro de software.

Distribuição: distribuição do código para terceiros;

Modificação: modificação do código pelo licenciado;

Uso comercial: se o código modificado pode ser utilizado comercialmente ou se deve ser compartilhado com a comunidade;

Sub-licenciamento: se o código modificado pode ser licenciado sob uma licença diferente (*e.g.*, um *copyright*) ou se deve reter a mesma licença sob a qual o mesmo foi disponibilizado.

Inclusão de *copyright*: se o código derivado deve incluir ou não a notificação de *copyright* do código originário.

Tabela 2.3. Exemplos de licenças de software de código aberto.

SPDX-id*	Licença
Apache-2.0	Licença Apache, versão 2.0
EPL-2.0	Licença Eclipse Public, versão 2.0
MIT	MIT License
GPL-3.0-only	Licença GNU Geral Pública, versão 3.0
LGPL-3.0-only	Licença GNU Geral Pública Atenuada v3.0
GPL-2.0+	Licença GNU Geral Pública, versão 2.0 ou superior
AGPL-3.0-only	Licença Affero GNU Geral Pública, versão 3.0
CC-BY-ND-4.0	Creative Commons Internacional com atribuições e sem modificações, versão 4.0

*Lista completa de identificadores SPDX discponível em `https://spdx.org/licenses/`

Royalties: se o cógigo derivado está livre do pagamento de *royalties* ou não.

Comunicar mudanças: se o código derivado deve informar explicitamente as mudanças realizadas no software originário.

Como pode ser visto na Tabela 2.4, mesmo com poucos atributos já é possível identificar elementos que distinguem as licenças consideradas na comparação. Embora todas as licenças consideradas concedam a permição para modificação e redistribuição do código, existem diferenças especialmente no tocante à possibilidade de sub-licenciamento e quanto a liberdade de *royalties*. Por exemplo, definir licença de um software como sendo EPL-*2.0*, implica dizer que aquele que desejar utilizar o software, o alterando, não terá a obrigacão de comunicar explicitamente as mesmas.

De maneira alguma, esta Tabela deve ser considerada como única referência na hora da escolha de uma licença para o seu software, uma vez que esta considera um número muito pequeno de atributos. No entanto, é possível comparar os mais de 40 atributos de diversas licenças com o auxílio de ferramentas *on-line* como o *Assistente de Licenciamento Joinup* disponibilizado pela *Comissão Européia*[3].

2.6. Indústria Criativa e Perspectivas Futuras

A partir dos anos 90 e a partir do uso massivo da Internet, surgiram termos relacionados a exploração econômica da atividade criativa nos mais diversos aspectos da economia. Assim, a economia e a indústria criativa ganharam força de impulsionar as atividades comerciais no mundo. Surgido na Austrália, em 1994, no entanto, difundido na Inglaterra [Bendassolli et al. 2009] a economia criativa foi um marco no crescimento econômico

[3]Disponível em: `https://bit.ly/3j8w1pz` *(Acesso em 21 de Agosto de 2021.)*

Tabela 2.4. Comparativo de permissionamento de licenças comumente utilizadas em software.

Licença	A	D	M	UC	SL	C	R	CM
Apache-2.0	☑	☑	☑	☑	☑	☑	☐	☑
EPL-2.0	☑	☑	☑	☑	☑	☑	☑	☐
MIT	☑	☑	☑	☑	☑	☑	☐	☐
GPL-3.0-only	☑	☑	☑	☑	☐	☑	☐	☑
LGPL-3.0-only	☑	☑	☑	☑	☐	☑	☐	☑
GPL-2.0+	☑	☑	☑	☑	☐	☑	☐	☑
AGPL-3.0-only	☑	☑	☑	☑	☐	☑	☐	☑
CC-BY-ND-4.0	☑	☑	☑	☐	☐	☑	☐	☐

A: Associação; **D**: Distribuição; **M**: Modificação; **UC** Uso Comercial; **SL**: Sub-licenciamento; **C**: Inclusão obrigatória de *copyright*; **R**: Livre de *royalties*; **CM**: Comunicar mudanças.

para essa região. O pioneirismo e associação deste tema com políticas públicas e econômicas na Inglaterra foi essencial para o desenvolvimento de mecanismos de produção e exploração comerciais. Entende-se por economia criativa, o conjunto de negócios baseados no capital intelectual e cultural e na criatividade que gera valor econômico. A indústria criativa, por outro lado, refere-se ao setor do mercado que estimula a geração de renda, cria empregos e produz receitas de exportação [Asato et al. 2019].

A indústria criativa surgiu, portanto, em consequência à quarta revolução industrial, também chamada de indústria 4.0, que tem como uma das principais características a incorporação da digitalização à atividade industrial, integrando tecnologias físicas e virtuais. Atividades como a exploração de *big data*, robótica avançada, computação em nuvem, inteligência artificial, sistemas de conexão máquina-máquina, sensores, atuadores e *softwares* de gestão avançada da produção foram incorporadas às atividades produtivas de alto rendimento nas tecnologias centrais de investimento [Russo et al. 2017].

A indústria criativa explora o potencial humano de manifestar-se mediante atividades transformadoras de bens imateriais em produtos tangíveis, por meio da utilização de recursos pessoais, desejos e suas fantasias. Esses conceitos levaram a inserção de cultura na economia, ou seja, a cultura e a criatividade exploradas enquanto bens de valor de mercado. Assim, foi possível estabelecer escalas de impactos dos movimentos culturais em vários setores da economia, tal como na Figura 2.7 do relatório da Conferência das Nações Unidas sobre Comércio e Desenvolvimento (CNUCD) [UNCTD 2012, Asato et al. 2019].

Os depósitos de patente e de registro de programas de computador estão diretamente ligados à indicação de avanço tecnológico dos países uma vez que correspondem a grande parte do incentivo à produção de tecnologias de fronteiras. Um mapeamento

Figura 2.7. Interconexões entre indústrias criativas *Fonte: [Asato et al. 2019]*.

mais recente da CNUCD [Secretariat 2021] mostrou que a economia movimentada por estas tecnologias foi em torno de 350 milhões de dólares e pode chegar a 3,2 trilhões até 2025. Por esse motivo os países devem promover a validação dessas de mecanismos de adaptação para esse novo paradigma tecnológico que sinaliza nos próximos anos. Para se preparar para esse movimento é importante estabelecer medidas em áreas específicas da indústria e economia criativa como: implantação de tecnologias da Informação e Comunicação, habilidades, atividade de pesquisa e desenvolvimento, atividade da indústria e acesso ao financiamento.

O panorama industrial e tecnológico no Brasil segue caindo em relação ao desenvolvimento mundial. Atualmente encontra-se no 62º no Índice Global de Inovação, produzido pela Universidade de Cornell, INSEAD e OMPI. Mesmo dentre os 18 países latino-americanos, o Brasil aparece na quarta posição, bem atrás do Chile (46º no Global), líder regional, Costa Rica, México, Panamá, Colômbia e Uruguai. Mais grave ainda, o país vem caindo no *ranking*: em 2011 ocupava a 47ª posição e caiu para a 69º em 2016 e 2017. Essa situação pode ser explicada por um conjunto de fatores sociais, políticos e econômicos [Carvalho et al. 2018]. A conjuntura econômica, queda no PIB, contração intensa da produção, aumento da inflação, desemprego e perda do poder de compra da nossa moeda causou um importante impacto sobre a indústria criativa nos últimos 10 anos. A alta burocracia, taxas de juros e baixo investimento em inovação, bem como a cultura pouco empreendedora nacional geram um ambiente inadequado ao desenvolvimento tecnológico e pouco adaptado às tendências internacionais.

Para superar esses desafios o Brasil precisa prioritariamente aumentar as taxas de crescimento econômico e da produtividade do trabalho. Um abrangente estudo incluindo dados de 151 países entre 1967 e 2011 mostrou que a renda dos 40% mais pobres aumenta de forma proporcional ao crescimento total da economia, confirmando a importância central das taxas de crescimento para a melhoria do bem-estar dos mais vulneráveis. Países que têm níveis de bem-estar mais altos tendem a ser precisamente aqueles que têm maior produtividade. Além disso, no mesmo sentido, estão intimamente relacionados ao cresci-

mento da indústria criativa e seus ativos, influenciados pelo desenvolvimento econômico e sustentado pelos investimentos de pesquisa e desenvolvimento. Atualmente, os maiores detentores de patentes no Brasil são as Universidades, onde os pesquisadores acadêmicos são os pioneiros no incentivo de P&D [Peixoto and Buainain 2021].

Contudo, não basta que apenas uma parcela da população se envolva nesse processo de crescimento. É importante o envolvimento maciço. Para que isso aconteça, se faz necessário o fomento a estratégias de empreendedorismo social, no sentido de engajar a população a respeito da necessidade de movimentar a economia, principalmente no que diz respeito a sua capacidade produtiva e criativa.

Referências

[Alvares et al. 2019] Alvares, H., Coelho, A. C., and Engel, M. S. P. (2019). *Manual do Usuário para o Registro Eletrônico de Programas de Computador*. Instituto Nacional da Propriedade Industrial (INPI), Rio de Janeiro – RJ. v1.8.5.

[Asato et al. 2019] Asato, T. A., Marques, H. R., Buzarquis, R. M., and Borges, P. P. (2019). Perspectivas da economia criativa e do desenvolvimento local no corredor bioceânico. *Interações (Campo Grande)*, 20(1):193–210.

[Bagnato et al. 2016] Bagnato, V. S., de Souza, M. A., and Murakawa, L. S. G. (2016). *Guia Prático I: Introdução à Propriedade Intelectual*. AUSPIN- Agência USP de Inovação.

[Bendassolli et al. 2009] Bendassolli, P. F., Jr., T. W., Kirschbaum, C., and e Cunha, M. P. (2009). Indústrias criativas: Definição, limites e possibilidades. *Revista de Administração de Empresas – RAE*, 49(1):10–18.

[Brasil 1996] Brasil (1996). Lei nº 9.279, de 14 de maio de 1996. *Diário Oficial da República Federativa do Brasil*.

[Brasil 1998a] Brasil (1998a). Decreto nº 2556, de 20 de abril de 1998. *Diário Oficial da República Federativa do Brasil*.

[Brasil 1998b] Brasil (1998b). Lei nº 9609, de 19 de fevereiro de 1998. *Diário Oficial da República Federativa do Brasil*.

[Carvalho et al. 2018] Carvalho, Z. V., da Silva Lima, E., da Silva, J. M. M., da Silva Ferreira, J., and da Costa Filho, L. A. (2018). *Incentivo à criatividade, pesquisa e desenvolvimento no ambiente produtivo – um guia de boas práticas de política de gestão em ciência, tecnologia e inovação para as empresas brasileiras*, pages 264–272. AAPI, Aracaju, SE.

[da Silva et al. 2021] da Silva, E. F., da Silva Borges, E. S., da Rocha Porto, P. C., and Peralta, P. P. (2021). *Patente: da importância e sua proteção: patente de invenção e modelo de utilidade*. INPI.

[de Mello e Souza et al. 2014] de Mello e Souza, A., Zucoloto, G. F., and da Rocha Porto, P. C. (2014). *Desafios atuais da proteção da propriedade industrial no brasil*, page 23p. IPEA, Brasilía.

[dos Deputados 2010] dos Deputados, C. (2010). *Legislação Brasileira sobre Direitos Intelectuais*. Edições Câmara, 4 edition.

[INPI 2013] INPI (2013). *A criação de uma marca: uma introdução às marcas de produtos e serviços para as pequenas e médias empresas*. INPI.

[Jungmann 2010] Jungmann, D. d. M. (2010). *A caminho da inovação: proteção e negócios com bens de propriedade intelectual: guia para o empresário*. IEL.

[Kappeler 2005] Kappeler, C. (2005). Histórico da propriedade intelectual: Como surgiu a propriedade intelectual no mundo e sua importância. https://www.direitonet.com.br/artigos/exibir/2113/Historico-da-Propriedade-Intelectual. Accessado: 26-08-2021.

[Peixoto and Buainain 2021] Peixoto, M. and Buainain, A. M. (2021). *Desempenho e Desafios do Sistema de Propriedade Industrial no Brasil*. Brasília: Núcleo de Estudos e Pesquisas/CONLEG/Senado. (Texto para Discussão n^o 294). Disponível em: <www.senado.leg.br/estudos>.

[Russo et al. 2017] Russo, S. L., de Morais Chaves Santos, M. R., Priesnitz, M. C., and Marques, L. G. A. (2017). *Propriedade intelectual, tecnologias e empreendedorismo*. Associação Acadêmica de Propriedade Intelectual (API).

[Secretariat 2021] Secretariat, U. (2021). Tecnology and innovation report. Technical Report UNCTAD/TIR/2020, United Nations, New York.

[Taddei 2010] Taddei, M. G. (2010). Marcas e patentes: os bens industriais no direito brasileiro. https://bit.ly/3kcD9BD. Accessed: 2021-08-28.

[UNCTD 2012] UNCTD, S. (2012). Trade and developmnt report. Technical Report UNCTAD/TDR/2012, United Nations, New York.

Capítulo 3

Uma Abordagem Prática para Aprendizagem em Arquitetura e Organização de Computadores com Apoio do Simulador Computacional CompSim

Guilherme Álvaro Rodrigues Maia Esmeraldo (IFCE), Eduardo Carlos Pereira da Silva Proto (IFCE), Edson Barbosa Lisboa (IFS) e Edna Natividade da Silva Barros (UFPE)

Abstract

This work presents a teaching-learning approach supported by a simulation environment for the discipline of Computer Architecture and Organization. In this chapter, initially, the discipline and its particularities are presented, justifying the need for simulation to support practical learning in design of computational systems. Following, the main concepts of Computer Organization and Architecture are worked on in the context of CompSim, a simulation environment with integrated graphical resources that simplify the design of new virtual (developed in software) or mixed (which can interact with hardware using Arduino platforms) computational systems.

Resumo

Este trabalho apresenta uma abordagem de ensino-aprendizagem com suporte de ambiente de simulação para a disciplina de Arquitetura e Organização de Computadores. Neste capítulo, inicialmente, apresenta-se a disciplina e suas particularidades, justificando a necessidade de simulação para apoiar o aprendizado prático em projetos de sistemas computacionais. Na sequência, são trabalhados os principais conceitos de Organização e de Arquitetura de Computadores no contexto do CompSim, um ambiente de simulação com recursos gráficos integrados que simplificam o projeto de novos sistemas computacionais virtuais (desenvolvidos em software) ou mistos (que podem interagir com hardware físico utilizando plataformas Arduino).

3.1. Introdução

O computador é um sistema de alta complexidade, composto de hardware e software. Devido à alta escala de integração, os subsistemas que compõem o hardware podem incluir bilhões de componentes eletrônicos menores e elementares, tornando-se exponencialmente complexo, o que impacta no seu estudo e entendimento. Assim, didaticamente, o computador pode ser definido e estudado de diferentes maneiras, como, por exemplo, em: 1) Stallings (2010), que cita que o computador é um sistema eletrônico que possui uma estrutura com subsistemas interrelacionados, e que cada subsistema também pode ser subdividido em novos subsistemas, formando uma estrutura hierárquica. Dessa forma, pode-se estudar e projetar cada subsistema independentemente e em momentos distintos, considerando sua estrutura e funções que serão desempenhadas; e 2) Tanenbaum e Austin (2013), que consideram que o computador é estruturado em camadas, e que, em cada camada, há uma linguagem específica para a programação do computador. Nesse contexto, entende-se que a linguagem de uma determinada camada depende da linguagem da sua camada antecessora. Assim, um programador que cria um programa de computador utilizando a linguagem de uma determinada camada, não precisa se preocupar com tradução para linguagens das camadas subjacentes.

Analisando as duas abordagens apresentadas, pode-se concluir que, de uma forma geral, um computador é um sistema eletrônico complexo com funções programáveis, e, dependendo da abordagem metodológica empregada, o seu estudo e aprendizado pode ser impactado.

3.1.1. Arquitetura e Organização de Computadores

Arquitetura e Organização de Computadores (AOC) é uma disciplina presente em cursos técnicos e superiores nas áreas de Computação e Engenharias Elétrica, Eletrônica e Mecatrônica, Automação Industrial, dentre outras [SBC 2005][ACM and IEEE 2013]. A disciplina inclui o estudo dos componentes do computador, das suas funções e dos modelos de comunicação, bem como dos aspectos visíveis ao programador [Stallings 2010]. Esses conhecimentos são fundamentais à operação, programação, projeto e otimização de desempenho de sistemas computacionais, além de estarem alinhados com as novas tendências tecnológicas, tais como: Internet das Coisas (IoT), Indústria 4.0, Smart Cities, Robótica, Computação de Alto Desempenho, entre outras.

Arquitetura de computadores trata dos aspectos visíveis ao programador, que são os aspectos que tratam da execução lógica dos programas de computador. Já a Organização de computadores trata dos componentes do computador, suas estruturas e submódulos, suas funções e as formas de comunicação entre si [Stallings 2010]. É importante observar que os conteúdos trabalhados na disciplina variam de acordo com o perfil de cada curso. Por exemplo, em cursos de graduação em Engenharia da Computação, há necessidade de um maior aprofundamento teórico-prático, o que já não ocorre em cursos de Sistemas de Informação.

As diretrizes curriculares da ACM e IEEE [ACM and IEEE 2013] incluem habilidades práticas que devem ser necessariamente desenvolvidas pelos estudantes, com objetivo de promover o aprofundamento teórico e permitir que os estudantes

possam explorar as características dos sistemas mais modernos [Nikolic et al. 2009]. O desenvolvimento das habilidades práticas pelos estudantes em AOC necessita de laboratórios de hardware especializados, com disponibilidade de componentes de hardware, ferramentas para manuseio e instrumentos de medição de diferentes grandezas em componentes eletrônicos, entre outros. Percebe-se assim que, compor e manter um laboratório desse porte exigirá maiores recursos financeiros e não é uma tarefa simples, pois necessitará de um projeto estrutural bem elaborado, de apoio de técnico de laboratório para preparação e acompanhamento dos experimentos, de manutenção dos equipamentos e de reposição dos componentes eletrônicos.

Este capítulo tem como objetivo apresentar uma abordagem prática de ensino-aprendizagem em AOC com suporte do simulador CompSim [CompSim 2021]. O Simulador CompSim é um ambiente virtual que inclui as principais características dos simuladores da literatura [Esmeraldo et al. 2019] e recursos gráficos integrados que permitem criar, programar, simular, visualizar e avaliar o desempenho de novos sistemas computacionais. Além disso, o CompSim traz suporte para integração do ambiente de simulação com a plataforma de prototipação Arduino, o que permite a criação de projetos de sistemas computacionais mistos (que envolvem tanto software quanto hardware físico). Este suporte minimiza a necessidade de uso de laboratórios especializados, tornando o simulador CompSim uma ferramenta com grande potencial para exploração e aplicação prática dos conceitos estudados na disciplina de AOC.

O uso de simuladores computacionais, ou ambientes virtuais, como prática pedagógica complementar, não é uma atividade nova [Balamuralithara and Woods 2009]. Estudos, como os apresentados em [Uribe et al. 2016] [Garcia, Pacheco and Garcia 2014] [Balamuralithara and Woods 2009], mostram que, ao se utilizar ambientes virtuais para fins educacionais, foi possível aumentar o desempenho acadêmico em cursos de tecnologia e engenharia. Os simuladores são ferramentas importantes no processo de apropriação do conhecimento, pois possibilitam o desenvolvimento de habilidades e experiências práticas, de forma assíncrona, em cenários virtuais que se assemelham aos reais [Wolffe et al. 2002]. São também fundamentais para compor laboratórios específicos na ausência de infraestrutura [Garcia, Pacheco and Garcia 2014] ou ainda quando há necessidade de se reduzir custos, realizar configurações rápidas e obter resultados instantâneos [Uribe et al. 2016]. Por fim, os simuladores são particularmente úteis para representação ou abstração de cenários complexos [Bahk et al. 2013] e frequentemente abordam os conteúdos presentes no estado da arte [Wolffe et al. 2002].

3.1.2. Uma Proposta de Abordagem Metodológica

Considerando as diferentes ementas, contextos locais e cursos de computação, a disciplina de AOC pode sofrer variações quanto à extensão e verticalização de seus conteúdos. Desta forma, sugere-se a leitura do trabalho em [Lisboa et al. 2019], o qual apresenta uma proposta de metodologia de ensino-aprendizagem que aborda, de forma flexível, um subconjunto de conteúdos comuns aos cursos de AOC, tais como: introdução à aritmética computacional, componentes do computador e suas funções, conjunto de instruções do processador, modos de endereçamento, entrada/saída, programação em baixo nível e análise de desempenho.

3.1.3. Conteúdo Programático

Este capítulo apresenta o simulador CompSim como um ambiente integrado com diferentes recursos para dar suporte aos processos de ensino-aprendizado prático em AOC. O capítulo está dividido da seguinte maneira:

Seção 3.2 - Introdução ao Simulador CompSim: esta seção apresenta brevemente o CompSim, destacando os principais recursos para apoio ao aprendizado em AOC;

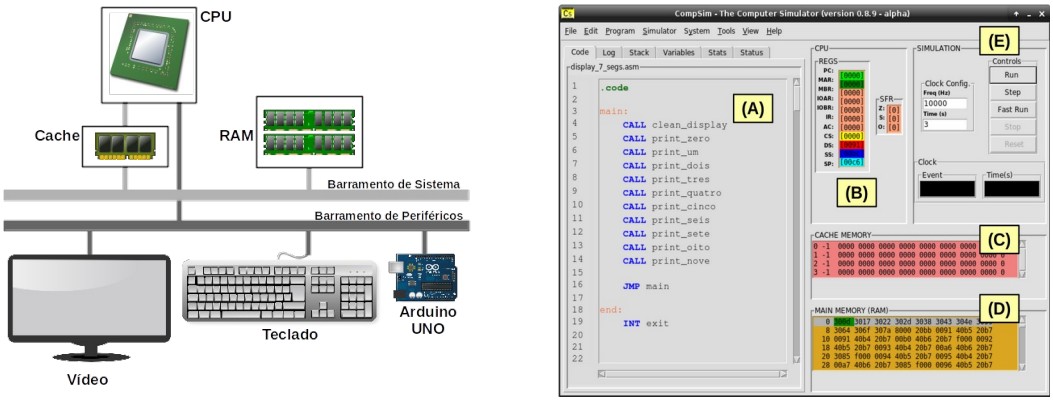
Seção 3.3 - Organização de Computadores com o CompSim: nesta seção, realiza-se uma breve introdução à organização de computadores, destacando as características dos principais componentes do computador (Processador, memórias RAM e Cache, Barramento e Subsistema de Entrada/Saída) no contexto do CompSim;

Seção 3.4 - Arquitetura de Computadores com o CompSim: esta seção trata de aspectos de arquitetura de computadores, tais como: modelo de memória e estrutura de um programa de computador; alocação e manipulação de dados em memória; processamento de dados; controle de fluxo de execução; modos avançados de acesso à memória; entrada/saída de dados; e modularização de programas; e

Seção 3.5 - Conclusões: Por fim, serão abertos espaços para discussões acerca do tema e de novos trabalhos.

3.2. Introdução ao Simulador CompSim

CompSim consiste de um ambiente virtual para apoio ao ensino-aprendizado em AOC [Esmeraldo and Lisboa 2017]. Ele segue a abordagem de projetos baseados em plataforma [Keutzer et al. 2000], na qual há uma Plataforma de Hardware Virtual (PHV) simulável e customizável, baseada em microprocessador, que inclui os principais componentes do computador, tais como processador, memórias, barramentos e periféricos, como pode ser vista na Figura 3.1(a).



(a) PHV do CompSim. **(b) Interface Gráfica do CompSim.**

Figura 3.1. PHV e Interface Gráfica do Simulador CompSim.

O simulador também conta com uma interface gráfica para dar suporte à configuração dos componentes da PHV, ajustar parâmetros de simulação e suportar a codificação de aplicações em baixo nível (linguagem Assembly). A Figura 3.1(b) mostra a interface gráfica do CompSim.

Na Figura 3.1(b) pode-se visualizar os seguintes componentes gráficos: A) Editor de código: inclui recursos para simplificar a codificação de uma aplicação, como número de linhas, teclas de atalho para recursos de edição (*copy*, *cut*, *paste*, *undo*/*redo*, *got to line*, etc.), um assistente de codificação (permite auxiliar a construção de instruções de código com a sintaxe correta), destaque de palavras-chave (*syntax highlight*), entre outros. Este componente está integrado a um montador (Assembler) que realiza análises léxica, sintática e semântica no código-fonte da aplicação, tradução deste para *bytecodes*, carregamento dos *bytecodes* na memória RAM, além de gerar um relatório do programa, que inclui a tabela de símbolos, endereços de memória dos segmentos e *bytecodes* gerados; B) Processador: durante uma simulação, exibe os registradores do processador e respectivos valores assumidos. Os registradores de endereçamento possuem cores diferenciadas, onde as respectivas cores são utilizadas para indicar as diferentes posições referenciadas na memória RAM; C) Memória cache: exibe as linhas da cache e respectivos valores, destaca também as linhas e as palavras endereçadas pelo processador durante uma simulação; D) Memória RAM: exibe os conteúdos (instruções e dados) de todos os endereços do componente virtual memória RAM (os componentes gráficos Memória RAM e processador estão vinculados, de forma que, durante uma simulação, as posições de memória são destacadas de acordo com as respectivas cores dos registradores de endereçamento); e E) Componentes de controle de configuração e execução de simulação: inclui controles que permitem configurar o tempo total de simulação e a frequência de relógio de sistema (*clock*), iniciar, executar (em modos normal, passo a passo ou rápido), parar e reiniciar uma simulação. Além desses, o CompSim conta com recursos para: registro de *logs* de simulação, acompanhamento dos status da pilha e variáveis do programa, geração de gráficos estatísticos após uma simulação, conversor de números inteiros, com e sem sinal, entre bases decimal, hexadecimal e binária, conversor de código-caractere ASCII, entre outros.

O simulador CompSim pode ser obtido no website do projeto [CompSim 2021], na seção "*Download*", e é distribuído para os sistemas operacionais MS/Windows e GNU/Linux. Após o download, a instalação consiste apenas em descompactar o arquivo obtido.

3.3. Organização de Computadores com o CompSim

Um computador é um dispositivo eletrônico que desempenha quatro funções principais: processamento, armazenamento e transferência de dados, bem como controle de operações, que envolve a coordenação das demais funções para permitir que os computadores realizem tarefas variadas [Stallings 2010]. Um computador é dividido em diferentes subsistemas, ou componentes, onde cada um deles possui uma estrutura e comportamento bem definidos. Assim, esses componentes devem se comunicar para trocar informações visando compor e realizar as funções do computador.

As subseções a seguir trazem as características básicas dos principais componentes do computador, que são Processador, Memória Principal, Memória Cache, Barramento e Subsistema de Entrada/Saída (E/S), no contexto dos componentes da PHV do CompSim.

3.3.1. Processador

O processador (*Central Processing Unit* - CPU) é o elemento central de qualquer computador. Ele é responsável por buscar instruções na memória, decodificá-las para compreender as tarefas que devem ser realizadas e executá-las [Tanenbaum and Austin 2013]. Além disso, ele interage com todos componentes do computador, dentre eles a memória e os periféricos. A CPU é o componente mais complexo do computador e, portanto, possui diversos subsistemas.

O CompSim inclui uma CPU, denominada de processador "Cariri", que apresenta as principais características de processadores reais. Dentre suas características, pode-se destacar:

- Possui arquitetura de 16-bits baseada em acumulador, onde, na maioria das suas operações, utiliza-se implicitamente o registrador de propósito geral, chamado "Acumulador" (AC). Por exemplo, em uma operação de adição, que necessita de dois operandos, a instrução deve referenciar apenas um dos operandos, pois considera-se que o outro operando já está contido em AC. Arquiteturas baseadas em acumulador são mais simples de projetar e programar, pois as instruções do processador possuem tamanhos e complexidade reduzidos [Null and Lobur 2009];
- Possui uma Unidade Lógica e Aritmética (ULA), responsável por realizar o processamento de dados, e uma Unidade de Controle (UC), que é responsável por coordenar todas as operações do processador;
- Possui um circuito Contador, que é um componente utilizado para incrementar automaticamente os endereços das instruções, contidos no registrador Contador de Programa (*Program Counter* - PC), que serão buscadas pelo processador na memória principal. Com suporte do contador, as instruções do programa são buscadas e executadas de forma sequencial;
- Conta com um Banco de Registradores, para suportar diferentes ações do processador, tais como: busca na memória e decodificação de instruções, operações lógicas e aritméticas, operações de E/S, de acesso à memória e à pilha de programa para armazenamento de dados, entre outras;
- Possui espaço de endereçamento diferenciado para operações de E/S e de acesso à memória principal. É importante destacar que, em muitos processadores, há apenas um espaço de endereços para acesso à memória e aos periféricos;
- Conta com 16 instruções de baixo nível, para diferentes operações tais como: transferência de dados, operações aritméticas e lógicas, controle de fluxo de programa e de sistema. As instruções são de 16-bits, sendo que 4-bits são reservados para o código de operação, ou seja, determinam a operação da instrução, e os demais 12-bits são reservados para o operando da instrução;

- Suporta dois tipos de operandos: inteiro de 16-bits com sinalização (*signed int*) e caractere (*char* ou *byte*); e
- Suporta os modos de endereçamento de dados: 1) imediato: onde o operando está na própria instrução; 2) direto: onde o operando pode ser acessado pelo endereço de memória na instrução (há um acesso à memória para busca do operando); 3) indireto: a instrução contém um endereço de memória que aponta para um outro endereço de memória, que é o endereço do operando (há dois acessos à memória para busca do operando); 4) registrador: o operando está contido em um registrador; 5) implícito: não há necessidade de informar onde está o operando.

3.3.2. Memória Principal

A memória principal (*Random Access Memory* - RAM) é o componente do computador onde as instruções e os dados de um programa em execução estão temporariamente armazenados. Para a execução de um programa, o processador deve ler instruções e trocar dados com a memória RAM [Stallings 2010]. As memórias RAM frequentemente são organizadas em matrizes, que não necessariamente precisam ser quadradas, onde cada célula é utilizada para armazenamento de um dado [Tanenbaum and Austin 2013]. Desta forma, um endereço de memória RAM é dividido em duas partes, sendo uma para seleção da linha e a outra para a coluna da matriz de dados, de forma a tornar disponível uma célula de armazenamento para leitura ou escrita de dados.

Na memória principal do CompSim, o modelo de endereçamento proposto utiliza endereços de 12-bits, sendo que os 9 bits mais significativos são utilizados para endereçamento de linha e os 3-bits menos significativos para endereçamento de coluna. Com essa configuração, é possível endereçar até 512 (2^9) linhas de memória, onde cada linha possui 8 (2^3) colunas, totalizando assim 4096 (512 x 8) diferentes endereços. Como a memória do CompSim armazena palavras de 16-bits - o termo "palavra", ou "*word*", refere-se a um conjunto de bits ou bytes, que consiste da unidade de informação que pode ser armazenada, transmitida e/ou processada em um computador -, sua capacidade de armazenamento total é de 65.536 (512 x 8 x 16) bits, ou simplesmente 8 KBytes.

3.3.3. Memória Cache

Na execução de um programa, o processador busca instruções e dados na memória RAM. Algumas instruções, por exemplo, necessitam realizar mais acessos à memória para busca dos operandos. E, considerando que um acesso à memória RAM leva muito tempo (consome vários ciclos de relógio de sistema), comparado à taxa de execução do processador, percebe-se que os acessos podem degradar o desempenho do processador e, por consequência, do sistema. Em resumo, o processador atua em uma taxa de execução muito mais alta do que a taxa de entrega de dados da memória RAM, caracterizando assim um modelo de comunicação frágil (muitos autores utilizam o termo "gargalo de comunicação") e que tem grande impacto no desempenho do sistema [Monteiro 2007]. Buscando otimizar o desempenho de comunicação Processador/Memória RAM, os projetistas de computadores criaram um tipo de

memória de armazenamento temporário, localizada entre o processador e a memória RAM e que é fabricada com a mesma tecnologia do processador, chamada de Memória Cache. A memória cache tem como função acelerar a entrega de instruções e dados, minimizando assim os ciclos de espera pelo processador.

A memória cache tem uma organização diferente da memória RAM. Enquanto que a memória RAM inclui uma matriz para armazenamento de dados, a memória cache armazena os blocos transferidos da memória RAM em linhas. Como a capacidade de armazenamento de blocos da memória cache é bastante inferior à da memória RAM, cada linha da cache pode ser utilizada para armazenar diferentes blocos, em momentos distintos. A abordagem de escolha de armazenamento de determinado bloco de memória RAM em uma determinada linha da memória cache se chama Mapeamento. Para maximizar o desempenho do sistema pelo uso de memórias Cache, é importante que todas as linhas da Cache estejam ocupadas com blocos transferidos da memória RAM. Porém, dependendo do(s) programa(s) em execução, haverá necessidade de retirar blocos armazenados na memória cache para ceder espaço para novos blocos que estão sendo transferidos (esse processo é conhecido como Substituição). Ao modificar um dado na memória Cache, é necessário manter a coerência com respectivo dado presente na memória RAM, daí deve-se utilizar alguma Política de Escrita (ou Política de Atualização) em memória cache.

A memória cache do CompSim possui as seguintes características: 1) Número de linhas parametrizável: é possível configurar o número total de linhas da memória cache; 2) Bloco de 8 palavras: cada linha poderá armazenar 8 palavras. A memória RAM do CompSim armazena 8 palavras de 16-bits por linha da sua matriz de dados. Assim, uma linha da memória cache poderá armazenar os dados contidos em uma linha da memória RAM; 3) Técnicas de Mapeamento: são suportadas as técnicas de Mapeamento Direto, Associativo e Associativo por Conjunto; 4) Algoritmos de Substituição: são suportadas as técnicas de substituição Menos Recentemente Usado (*Least Recently Used* - LRU), Fila Circular (*First-In First-Out* - FIFO) e Aleatório (*Random*); e Políticas de Escrita: são suportadas as políticas de escrita com acerto *Write-Through* e *Write-Back*, e as políticas de escrita com falta *Write-Allocate* e *Write-Around*.

3.3.4. Barramento

Um barramento é um recurso utilizado para conectar os componentes do computador [Null and Lobur 2009] e possibilita a comunicação de dados e controle entre eles. Os barramentos compartilhados são subdivididos em três tipos: 1) Endereço: é utilizado para informar um endereço de algum componente para realizar algum tipo de comunicação; 2) Controle: é utilizado para informar o tipo de comunicação que será realizada com o componente endereçado no barramento de endereço; e 3) Dados: é utilizado para a transferência de dados entre os componentes comunicantes. As larguras, ou os números de linhas ou de sinais, dos barramentos de dados e endereços, são parâmetros importantes que têm grande impacto no desempenho de comunicação e capacidade de endereçamento do sistema computacional. A largura do barramento de dados define a quantidade de bits que podem ser transferidos em paralelo em uma única comunicação. Já a largura do barramento de endereços define a quantidade de unidades

endereçáveis, quer sejam o total de endereços de memória ou número de periféricos. Os barramentos podem ser classificados em Síncronos, que são aqueles onde a sequência de eventos de uma transmissão de dados (transação) é controlada por eventos gerados pelo relógio do sistema (*clock*); e Assíncronos, cujas linhas de controle coordenam a transmissão de dados através de um protocolo de comunicação (*handshaking*).

No CompSim, como o processador Cariri possui espaços de endereçamento diferenciados para comunicação com a memória e com os periféricos, a PHV inclui um barramento síncrono de sistema para conectar processador, memória cache e memória RAM, e um barramento assíncrono de periféricos para conectar o processador Cariri ao subsistema de E/S para se comunicar com os periféricos. O barramento de sistema possui largura de barramentos de dados de 16-bits (comunicações de 2 bytes) e de endereços de 12-bits (endereça até 4.096 posições de memória RAM) e suporta as operações de leitura e escrita de dados; enquanto que o barramento de periféricos possui larguras de barramentos de dados e de endereços de 8-bits (comunicações de 1 byte e endereçamento de até 256 periféricos) e também suporta operações de leitura e escrita.

3.3.5. Subsistema de E/S

Os periféricos são componentes fundamentais em computadores, pois são eles que possibilitam a interação entre o computador e o usuário (*Human Readable*), o computador e outros sistemas computacionais (*Machine Readable*) e comunicação entre dispositivos remotos (*Communication*). Cada tipo de periférico possui características próprias, tais como tecnologia de fabricação, funcionalidades, mecanismos de operação, taxas de transferência de dados, formato de dados e tamanhos de palavra [Stallings 2010]. O processador se comunica com os periféricos através do barramento de periféricos, o qual, por sua vez, não possui capacidade para realizar conexão com as interfaces nativas de cada um dos tipos de periféricos. Em outras palavras, é necessário que cada periférico, independentemente da sua natureza, inclua um mecanismo que permita sua conexão com uma interface padronizada presente no barramento de periféricos. Essa interface com o barramento de periféricos é conhecido como Subsistema de E/S, Módulo de E/S ou ainda Controlador do Periférico.

O simulador CompSim inclui um subsistema de E/S conectado ao barramento de periféricos da PHV. Esse subsistema de E/S permite que novos periféricos sejam conectados de forma automática ao barramento de periféricos, bastando que o periférico implemente a interface de comunicação padrão com o barramento. Quanto aos periféricos, é possível ter periféricos do tipo virtual (que são aqueles implementados em software e emulam o comportamento de periféricos reais), e do tipo físico (que são divididos em software, para implementar a interface padrão com o barramento, e em hardware para compor o periférico físico em si e seu comportamento).

3.4. Arquitetura de Computadores com o CompSim

A CPU é o componente mais importante do computador. Segundo Tanenbaum e Austin (2013), ela é considerada o "cérebro" do computador, pois sua função é executar os programas armazenados na memória RAM, buscando suas instruções, interpretando-as e executando-as uma a uma, de maneira sequencial.

Independentemente da linguagem de programação utilizada para criar os programas de computador, a CPU somente reconhece o seu próprio conjunto de instruções da arquitetura (*Instruction Set Architecture* - ISA). Assim, para que seja possível executar os programas criados em linguagem de alto nível pelos programadores, suas instruções devem ser traduzidas, por um compilador, para um novo programa que estará descrito em termos da ISA da CPU. Algumas CPUs possuem uma ISA reduzida, com instruções simples, de tamanho fixo e que requerem poucos ciclos de *clock* para serem executadas (abordagem *Reduced Instruction Set Computer* - RISC); já outras trazem um conjunto mais amplo, com instruções de tamanho variado e que requerem vários ciclos de *clock* para serem executadas (abordagem *Complex Instruction Set Computer* - CISC). A arquitetura RISC, por conter instruções mais simples, pode ser considerada mais rápida do que a arquitetura CISC, por outro lado, os programas tendem a conter mais instruções. Atualmente, dispositivos móveis, tais como *smartphones* e *tablets*, incluem processadores RISC. Já os computadores pessoais, tais como *desktop* e *notebook*, possuem processadores CISC.

A CPU do CompSim trata exclusivamente de palavras de 16-bits. Isso significa que suas instruções e os tipos de dados suportados possuem larguras de 16-bits. Em cada instrução, foram reservados 4-bits para o campo de código de operação ("*Opcode*") e 12-bits para o campo Operando. Com opcodes de 4-bits, a CPU suporta 16 (2^4) instruções, que podem ser dos tipos de transferência de dados, aritméticas, lógicas, entrada/saída, transferência de controle e controle de sistema. Ressalta-se que, apesar do processador Cariri oferecer uma ISA restrita (poucas instruções), suas instruções possuem versatilidade suficiente para: incluir duas ou mais operações em uma única instrução; suportar diferentes modos de endereçamento e, com isso, ampliar os recursos de manipulação de dados pelo processador; e suportar a composição de combinações de instruções para implementar outras instruções e/ou operações mais complexas. As instruções da CPU podem suportar um tipo de operando numérico ou um tipo de dado não numérico. O tipo numérico consiste de números inteiros de 16-bits com sinal (*signed int*), codificados em notação "Complemento a 2". Já o tipo não numérico consiste de caracteres (*char*), codificados no padrão ASCII.

As subseções a seguir apresentam alguns dos aspectos mais abordados no estudo de arquitetura de computadores, ilustrando-os com aplicações no simulador CompSim.

3.4.1. A Estrutura de um Programa

Em cada célula de dados da memória RAM, é possível armazenar uma palavra, que consiste de uma sequência de bits ou de bytes. As células podem ser utilizadas para armazenamento de diferentes tipos de dados e de instruções, desde que estejam codificados em bits.

Para diferenciar dados e instruções, bem como reservar mais recursos na memória RAM, normalmente, os programas de computador, antes de serem alocados em memória, são divididos em segmentos, tais como: 1) Código ou de Texto (*Code*/*Text*): inclui todas as instruções do programa; 2) Dados (*Data*): inclui todas as variáveis e estruturas de dados globais ou estáticas, que foram definidas (inicializadas com valores na sua criação); 3) BSS (*Block Started by Symbol*): inclui todas as variáveis e estruturas de dados globais ou estáticas, que foram apenas declaradas (não foram

inicializadas); 4) *Heap*: é um segmento que contém espaços de memória não utilizados, os quais podem ser demandados em operações de alocação dinâmica de memória, tais como pelo uso das instruções “malloc” e “calloc” da linguagem de programação C; 5) Pilha (*Stack*): este segmento é utilizado para implementar a pilha do programa, que é basicamente é uma estrutura de dados do tipo LIFO (*Last-In*, *First-Out*) e pode ser utilizada para armazenamento temporário de dados, passagem de parâmetros de entrada e de retorno em funções, implementação de variáveis locais, entre outros.

A CPU do CompSim conta com 4 registradores para dar suporte à segmentação da memória de um programa. São eles: 1) Registrador de Segmento de Código (*Code Segment* - CS): armazena o endereço da primeira posição de memória alocada para o segmento de código; 2) Registrador de Segmento de Dados (*Data Segment* - DS): armazena o endereço da primeira posição de memória alocada para o segmento de dados; 3) Registrador de Segmento de Pilha (*Stack Segment* - SS): armazena o endereço da primeira posição de memória alocada para o segmento de pilha; e 4) Registrador Apontador de Topo de Pilha (*Stack Pointer* - SP): aponta para o topo da pilha do programa. Com suporte dos registradores de segmentos, o processador Cariri tem ciência dos limites de cada um dos segmentos.

O código a seguir ilustra a estrutura de um programa em linguagem de baixo nível (*Assembly*), no CompSim.

Linha	Código
1	`.code`
...	...
...	`.data`
...	...
...	`.bss`
...	...
...	`.stack 10`

Na estrutura do código anterior, observa-se que ela inclui as palavras-chave “.code”, “.data”, “.bss” e “.stack”, que são delimitadores de seções em um código *Assembly* e indicam: o início da seção que conterá todas as instruções do programa; o início da seção que conterá a definição (declaração com inicialização) de variáveis e estruturas de dados globais e estáticas do programa; o início da seção que conterá a declaração (sem inicialização) das variáveis e estruturas de dados globais e estáticas do programa; e o tamanho da pilha, respectivamente. A palavra-chave “.stack” deve ser sucedida por um número inteiro positivo, o qual informa a quantidade de posições de memória que serão alocadas para a pilha do programa.

3.4.2. Alocação e Manipulação de Dados em Memória

No CompSim, a criação de variáveis e estruturas de dados é dada pelo uso de pseudo-instruções do montador (Assembler). Uma pseudo-instrução é um tipo de instrução que existe na linguagem *Assembly*, porém somente é reconhecida pelo montador (não faz parte do conjunto de instruções da CPU). Após o processo de montagem do código-fonte, durante o carregamento do programa em memória, o montador fica responsável

por alocar memória suficiente para comportar as estruturas de dados criadas com as respectivas pseudo-instruções no código-fonte.

As definições de variáveis dos tipos inteiro e caractere podem ser realizadas através das pseudo-instruções DD e DB, respectivamente. Já as definições de *arrays* dos tipos inteiro e caractere podem ser realizadas através das pseudo-instruções INITD e INITB, respectivamente. Neste capítulo, será utilizado o termo "*array*" para fazer referência à estrutura de dados matriz, quer seja uni ou multidimensional, visto que, do ponto de vista de alocação de memória, as linhas de uma matriz multidimensional são armazenadas sequencialmente em memória, assumindo assim uma disposição linear (vetorial). Um *array* de caracteres consiste, na realidade, em uma cadeia de caracteres (*string*), a qual, por sua vez, em termos de alocação em memória, apresenta-se como um *array* de números inteiros, visto que os caracteres da cadeia são codificados no padrão ASCII. É importante ressaltar que, na definição de variáveis e *arrays* do tipo inteiro, os valores de inicialização podem ser informados em bases decimal, hexadecimal e binária.

Uma variável, ou uma estrutura de dados, pode ser declarada e não possuir um valor de inicialização. As declarações devem ser realizadas exclusivamente na seção ".bss" do código *Assembly* do processador Cariri. Desta forma, o montador reconhece, na declaração, o tipo e a quantidade de elementos que a estrutura de dados necessita e, com isso, aloca espaços de memória suficientes para comportá-la. As declarações de variáveis e de *arrays* do tipo inteiro e caractere podem ser realizadas através da pseudo-instruções RESD e RESB, respectivamente.

No código a seguir, ilustra-se a criação de diferentes estruturas de dados.

Linha	Código
1	`.code`
...	`...`
8	`.data`
9	`...`
10	`   var1:   DD 10`
11	`   var2:   DB 'A'`
12	`   array1: INITD 10,11,12`
13	`   array2: INITB "ABC",0`
14	`.bss`
15	`   var3:   RESD 1`
16	`   array3: RESB 3`
17	`.stack 10`

No código anterior, pode-se visualizar: a definição de uma variável do tipo inteiro inicializada com o valor 10 ("var1" na Linha 10) e uma do tipo caractere com valor 'A' ("var2" na Linha 11); a definição de um *array* inicializado com os números inteiros 10, 11 e 12 ("array1" na Linha 12) e de um *array* de caracteres inicializado com a *string* "ABC\0" ("array2" na Linha 13). Nas Linhas 15 e 16, são declarados, respectivamente, a variável "var3" do tipo inteiro (reservou-se 1 espaço de memória para ela) e o *array* de caracteres "array3", para o qual foi reservado espaço para armazenamento de 3 caracteres.

A manipulação de dados é realizada na CPU. A CPU do CompSim possui instruções para realizar operações de leitura e de escrita de dados em memória. Nessas instruções, o acesso aos dados alocados em memória pode ser realizado a partir de um nome de uma variável ou de um *array*, ou ainda por um endereço de memória. As instruções para leitura e gravação de um dado em memória são LDA e STA, respectivamente.

O programa a seguir ilustra como pode ser realizada a atribuição de um valor de uma variável "var1" à uma variável "var2". Uma atribuição de valor entre variáveis consiste em realizar a leitura do valor de uma variável, em uma determinada posição de memória, pela CPU, e escrevê-lo na posição de memória referente à outra variável. No código, na seção ".data", a variável "var1" é definida na Linha 11, sendo inicializada com o inteiro 10. Já a variável "var2" é declarada na seção ".bss", na Linha 14, por não ser inicializada.

Linha	Código
1	`.code`
2	`   LDA var1          ;as duas instrucoes realizam: var2 = var1`
3	`   STA var2`
4	
5	`end:`
6	`   INT exit`
7	
8	`.data`
9	`   ;syscall exit`
10	`   exit: DD 25`
11	`   var1: DD 10       ;int var1 = 10`
12	
13	`.bss`
14	`   var2: RESD 1      ;int var2`
15	`.stack 10`

As instruções de um programa devem ser inseridas na seção ".code". Assim, no código anterior, a Linha 2 inclui a instrução LDA que implementa a operação de leitura do inteiro 10, contido na variável "var1" em memória, para o registrador AC (Acumulador). Já na Linha 3, a instrução STA realiza a gravação do dado contido no registrador AC para a posição de memória referente à variável "var2".

Ainda no código, as Linhas 5 e 6 apresentam um rótulo "end:" e a respectiva instrução "INT exit" (as instruções em linguagem *Assembly* podem conter rótulos, também chamados de *labels*, e são muito importantes para dar suporte ao controle de fluxo de execução, modularização do programa, entre outros). A instrução INT, que recebe, como operando, uma variável ou um endereço de memória, possui múltiplas funções, dependendo do valor lido em memória, como será visto ao longo deste capítulo. Na Linha 6, a instrução INT realiza a leitura do valor da variável "exit", a qual contém o inteiro 25 (a variável "exit" é definida na Linha 10). Uma instrução INT com parâmetro inteiro 25, quando decodificada pela CPU, será compreendida como a instrução de controle de parada de simulação (*Halt*). Com isso, ao simular essa aplicação, a instrução "INT exit" instrui a CPU do CompSim a encerrar sua execução.

3.4.3. Processamento de Dados

No computador, o processamento de dados é realizado na CPU ou, mais precisamente, na ULA da CPU. A ULA é um circuito do tipo combinacional, que dependendo da sua entrada, logo apresenta uma saída. A ULA deve dar suporte aos tipos de processamento de dados demandados pelas instruções presentes na ISA da CPU. Dependendo da instrução de processamento de dados, a UC decodifica a instrução e configura a ULA para realizar tal operação.

Geralmente, nas CPUs, é possível encontrar diferentes tipos de instruções para processamento de dados, sendo as mais comuns as operações aritméticas, tais como adição, subtração, multiplicação e divisão, bem como as operações lógicas, tais como os operadores lógicos AND, OR, NOT e XOR e operações de deslocamento de bits. A CPU do CompSim conta com apenas 4 instruções para processamento de dados, sendo 2 delas para operações aritméticas (adição - ADD e subtração - SUB), 1 lógica (negação da conjunção - NAND) e 1 para deslocamento aritmético de bits (SHIFT).

O código a seguir ilustra como pode-se implementar a instrução “var1 = var1 + var2”, em linguagem Assembly do CompSim. As variáveis “var1” e “var2” estão definidas nas linhas 12 e 13, e são inicializadas com os inteiros 10 e 20, respectivamente. Na Linha 2, é realizada a leitura do valor contido na posição de memória referente à “var1” para AC (“AC = var1”); na Linha 3 realiza-se a soma dos inteiros em AC e no endereço de memória referente à “var2”, sendo o resultado guardado no próprio AC (“AC = AC + var2”); por fim, na Linha 4, o inteiro em AC é gravado na posição de memória referente à variável “var1” (“var1 = AC’).

Linha	Código
1	`.code`
2	`   LDA var1          ;este bloco realiza: var1 = var1 + var2`
3	`   ADD var2`
4	`   STA var1`
5	
6	`end:`
7	`   INT exit`
8	
9	`.data`
10	`   ;syscall exit`
11	`   exit: DD 25`
12	`   var1: DD 10       ;int var1 = 10`
13	`   var2: DD 20       ;int var2 = 20`
14	`.stack 10`

No CompSim, as operações lógicas são implementadas a partir da instrução NAND. A operação lógica NAND é bastante versátil, pois, com ela, é possível compor outras operações e funções lógicas, utilizando algumas propriedades da álgebra booleana, por isso ela é conhecida como “porta lógica universal” [Patrick, Fardo and Chandra, 2020]. Para compreensão de como funções lógicas podem ser compostas a partir de operações lógicas universais, é importante ter conhecimentos básicos de

álgebra booleana. Por exemplo, a composição da operação lógica AND a partir de operações NAND, se dá da seguinte maneira:

$$A.B = \neg ((\neg (A.B)) . (\neg (A.B))) \quad (1)$$

, onde “A e “B” são variáveis lógicas, “¬” representa o operador lógico de negação; e “.” representa o operador lógico de conjunção. Partido da equação (1), o código a seguir ilustra como pode ser implementada uma operação lógica de conjunção (AND) entre as variáveis “var1” e “var2”.

Linha	Código
1	`.code`
2	`   LDA var1          ;este bloco realiza: tmp = NAND(var1, var2)`
3	`   NAND var2`
4	`   STA tmp`
5	
6	`   NAND tmp          ;realiza: tmp = NAND(AC, tmp)`
7	
8	`end:`
9	`   INT exit`
10	
11	`.data`
12	`   ;syscall exit`
13	`   exit: DD 25`
14	`   var1: DD 1111111111111111b      ;int var1 = 0xffff`
15	`   var2: DD 0000000011111111b      ;int var1 = 0x00ff`
16	`   tmp:  DD 0`
17	`.stack 10`

As variáveis “var1” e “var2”, que estão sendo definidas nas Linhas 14 e 15, são inicializadas com os inteiros em base binária “1111111111111111b” e “0000000011111111b” (65.535 e 255, respectivamente). Nas Linhas 2 à 4, realiza-se a operação lógica NAND entre os bits das variáveis “var1” e “var2”, e o resultado é armazenado na variável “tmp”, tendo assim, portanto: “tmp = ¬(var1 . var2)”. O próximo passo, considerando a equação (1), será realizar uma nova operação NAND com os bits em “tmp”. Assim, a Linha 6 realiza a operação lógica NAND entre os bits em AC e na variável “tmp” (neste caso, os bits em AC estão idênticos aos de “tmp”), tendo assim “AC = ¬(AC . tmp)”. O resultado desta operação, guardado em AC, contém o valor lógico equivalente ao da operação AND entre os bits das variáveis “var1 e “var2”.

A instrução lógica de deslocamento de bits SHIFT da CPU do CompSim realiza deslocamento aritmético de 1-bit em palavras de 16-bits, nos sentidos à direita ou à esquerda. Essa operação tem várias utilidades, sendo uma delas a implementação da operação de multiplicação por 2 (deslocamento à esquerda) ou divisão por 2 (deslocamento à direita). A combinação da instrução SHIFT com a de adição ADD pode ser utilizada para implementar outras multiplicações tal como, por exemplo, por 5.

Em tese, uma multiplicação de um inteiro 10 por 5, poderia ser implementada com adições sucessivas (10+10+10+10+10). Porém, com o uso de deslocamento de bits, 10 poderia ser multiplicado por 4 (através de dois deslocamentos sucessivos à esquerda), seguido de uma uma soma por 10, como mostra o código a seguir.

Linha	Código
1	`.code`
2	`   LDA var1          ;este bloco realiza: (var1*2)*2 + var1`
3	`   SHIFT esquerda`
4	`   SHIFT esquerda`
5	`   ADD var1`
6	
7	`end:`
8	`   INT exit`
9	
10	`.data`
11	`   ;syscall exit`
12	`   exit:      DD 25`
13	`   var1:      DD 10`
14	`   esquerda:  DD 1`
15	`.stack 10`

No código anterior, são definidas as variáveis “var1” (na Linha 13), que é inicializada com o inteiro 10, e a variável “esquerda” (na Linha 14), que é inicializada com o inteiro 1 (esta variável será utilizada pela instrução SHIFT e o inteiro 1 indica que será realizado um deslocamento de 1-bit à esquerda). Na Linha 2, o valor de “var1” é carregado em AC. As instruções SHIFT, nas Linhas 3 e 4, atuam da seguinte maneira: ao ler o valor da variável “esquerda”, verifica-se que consiste de um deslocamento de 1-bit à esquerda no valor contido em AC. Assim, com as instruções das Linhas 3 e 4, o valor em AC sofrerá 2 deslocamentos de 1-bit à esquerda, o que equivale à multiplicação por 4 (“AC = AC*2*2”). Por fim, a instrução na Linha 5, adiciona o valor de “var1” ao contido em AC, compondo a multiplicação por 5. Para realizar o deslocamento de 1-bit à direita, a instrução SHIFT, ao ler o conteúdo de uma variável, necessita do valor 0.

É importante destacar que a aplicação das operações lógicas e aritméticas pode resultar, além de um valor positivo, também pode resultar em valores zero, negativo e que supera o limite de 16-bits de uma palavra (*overflow*). Nesses casos, a CPU notifica-os por meio de um registrador de estados (*Status Flags Register* - SFR), que possui 1-bit para sinalizar resultado igual a zero (Z), 1-bit para resultado negativo (S) e 1-bit para *overflow* (O).

3.4.4. Controle de Fluxo de Execução

Um programa de computador consiste em um sequência de instruções, as quais são lidas e executadas respeitando uma ordem de execução. Um controle de fluxo de execução, em termos gerais, pode consistir em, dada uma determinada condição, alterar a ordem em que as instruções de um programa são executadas. Controle de fluxo pode ser importante em várias situações, tais como: dada uma condição, executar uma de duas

possíveis ações; repetir uma sequência de ações até que uma condição não mais seja atendida; repetir determinadas ações por uma quantidade predefinida de vezes; entre outras.

A CPU do CompSim possui três instruções de controle de fluxo, sendo uma delas de salto incondicional (JMP) e duas de salto condicional (JZ e JN). A instrução JMP (*Jump*), ao ser executada, transfere seu operando, que consiste de um endereço de memória, para o registrador contador de programa (PC), de forma que, no próximo ciclo de instrução da CPU, o programa passará a executar a instrução apontada no novo endereço em PC. Já as instruções de salto condicional, antes de atualizarem o endereço em PC, avaliam se determinados bits no registrador de status SFR estão configurados em 1. No caso, a instrução JZ (*Jump if Zero*) avalia o bit Z do registrador SFR, enquanto que a instrução JN (*Jump if Negative*) avalia o bit S de SFR. Por exemplo, se estiver sendo executada uma instrução JZ e se o bit Z de SFR estiver configurado em 1, a instrução fará com que o endereço de memória contido em seu operando seja copiado para PC. Dessa forma, no próximo ciclo de instrução da CPU, será executada a instrução apontada pelo novo endereço em PC. Após um salto condicional, as instruções JZ e JN configuram em 0 os bits Z e S de SFR, respectivamente. Com o uso das instruções JMP, JZ e JN é possível implementar construções de linguagens de alto nível para controle de fluxo, tais como IF..ELSE, FOR..DO e WHILE..DO.

O código a seguir ilustra a implementação de uma estrutura IF..ELSE, na qual verifica-se se o valor da variável “var1” é superior ao da variável “var2”. Dependendo do resultado da comparação, será atribuído o conteúdo da variável de maior valor à de menor valor.

Linha	Código
1	`.code`
2	`if:`
3	`   LDA var1          ;este bloco realiza: if (var1 >= var2)`
4	`   SUB var2`
5	`   JN else`
6	`then:                ;este bloco realiza: then`
7	`   LDA var1             ;var2 = var1`
8	`   STA var2`
9	`else:                ;este bloco realiza: else`
10	`   LDA var2             ;var1 = var2`
11	`   STA var1`
12	
13	`end:`
14	`   INT exit`
15	
16	`.data`
17	`   ;syscall exit`
18	`   exit: DD 25`
19	`   var1: DD 10`
20	`   var2: DD 20`
21	`.stack 10`

No código, para a comparação, as Linhas 3 e 4 realizam a subtração entre os valores contidos nas variáveis “var1” e “var2”. Como o valor da variável “var1” é inferior ao da variável “var2” (ver suas definições nas Linhas 19 e 20), a CPU configurará em 1 o bit S no registrador SFR. Seguindo o fluxo de execução, na Linha 5, a instrução JN verificará que o bit S de SFR está configurado em 1, então, ela atribuirá o endereço da instrução rotulada como “else” ao registrador PC, provocando assim, um desvio do fluxo para essa instrução, no próximo ciclo de execução da CPU.

3.4.5. Acesso Avançado à Memória

Viu-se, nas subseções anteriores, que a CPU possui recursos que permitem realizar o acesso e o processamento de dados armazenados em memória. Dependendo do conjunto de dados e do modelo de processamento da aplicação, pode ser necessário utilizar mecanismos adicionais para ampliar as formas de acesso ao conjunto de dados e aos seus elementos individualmente, para, desta forma, reduzir a complexidade e otimizar a aplicação.

Um desses recursos é o ponteiro. Um ponteiro, ou apontador, pode ser definido como uma variável que armazena números inteiros sem sinal com dimensão suficiente para acesso a todos os endereços de memória do espaço de endereçamento da CPU [Santos and Langlois 2018]. Os ponteiros podem ser utilizados em diversas situações, tais como: referenciar variáveis, provendo um caminho alternativo para acesso aos dados das variáveis; acesso aos elementos de estruturas de dados com armazenamento contínuo em memória, como *arrays*, e não alocadas em endereços contíguos de memória, tais como árvores e grafos; passagem e retorno de parâmetros por referência em funções; entre outras.

A CPU do Compsim possui instruções que permitem definir um ponteiro (DD), ler (LDI) e gravar (STI) o conteúdo apontado na memória, bem como atribuir o endereço de uma variável a um ponteiro (MOV). O código a seguir ilustra uma aplicação que utiliza cada uma dessas funções.

Linha	Código
1	`.code`
2	`   LDI ptr          ;este bloco realiza: *ptr = *ptr + var2`
3	`   ADD var2`
4	`   STI ptr`
5	
6	`   MOV var2         ;este bloco realiza: ptr = &var2`
7	`   STA ptr`
8	
9	`end:`
10	`   INT exit`
11	
12	`.data`
13	`   ;syscall exit`
14	`   exit: DD 25`
15	`   var1: DD 10`
16	`   var2: DD 20`
17	`   ptr:  DD var1    ;int * ptr = &var1`
18	`.stack 10`

No código, são definidas as variáveis “var1” e “var2”, nas Linhas 15 e 16. Na Linha 17, define-se um ponteiro, chamado “ptr”, que é do tipo inteiro, e é inicializado com o endereço de memória da variável “var1” (equivalente à instrução em linguagem C “int * ptr = &var1”). Na Linha 2, o conteúdo apontado por “ptr” é lido para AC, através da instrução LDI (“AC = *ptr”). Na Linha 3, é acrescido ao conteúdo lido o valor da variável “var2” (“AC = AC + var2”). Na Linha 4, o resultado da operação de adição anterior é gravado na posição de memória apontada por “ptr”, através da instrução STI (“*ptr = AC”). Por fim, nas Linhas 6 e 7, o endereço de memória da variável “var2” é copiado para AC (“AC = &var2”) e, em seguida, é gravado na variável “ptr” (“ptr = AC”), fazendo com que a mesma passe a apontar para a posição de memória referente à “var2”.

Com o suporte das instruções de alocação e de manipulação de dados em memória, controle de fluxo de execução e de acesso avançado à memória, é possível criar programas mais elaborados, tais como para manipulação de estruturas de dados mais complexas, como matrizes, registros, listas encadeadas, árvores e grafos.

3.4.6. Entrada/Saída

A interação com o computador se dá através dos periféricos. Viu-se que há diferentes tipos de periféricos, onde cada tipo possui suas próprias características; e que a comunicação entre a CPU e os periféricos se dá pelo barramento de periféricos e pelo subsistema de E/S.

O subsistema de E/S do CompSim permite a conexão automatizada de novos periféricos ao barramento de periféricos, bastando que o periférico implemente a interface de comunicação padrão com o barramento. Quanto aos periféricos, no CompSim, é possível ter periféricos do tipo virtual, que são aqueles implementados em software e emulam o comportamento de periféricos reais, e físico, que são divididos em software, para implementar a interface padrão com o barramento de periféricos, e em hardware para compor o periférico físico em si e seu comportamento. Por padrão, o CompSim não é distribuído com periféricos. Porém, no website do projeto [CompSim 2021], na seção “*Download*” é possível realizar o *download* para instalação dos periféricos virtuais Video, Keyboard e VArduino (Arduino Virtual) e do periférico físico Arduino UNO, tanto para o sistema operacional MS/Windows quanto GNU/Linux. A instalação de um periférico é bastante simples: consiste apenas em descompactar o arquivo referente ao periférico, obtido no website do CompSim, na pasta onde se encontra o arquivo executável do simulador CompSim.

A CPU do CompSim possui a instrução INT, a qual, até este ponto do capítulo, somente havia sido utilizada com o parâmetro 25 para encerrar a execução do programa (*Halt*). Porém, a instrução INT também pode ser utilizada em operações de E/S com periféricos. Para tanto, a instrução necessita, como parâmetros, dos códigos de operação 20 e 21 para leitura e escrita de dados em um determinado periférico, respectivamente. As operações de E/S, com a instrução INT, devem atender às seguintes condições: 1) Em operações de leitura/escrita, os 8-bits mais significativos do registrador acumulador AC serão utilizados para endereçamento de periféricos. Com isso, é possível ter até 256

endereços de periféricos (“portas”) diferentes (2^8 = 256); 2) Em operações de saída, os 8-bits menos significativos do registrador acumulador AC serão utilizados para guardar o byte que será escrito em algum periférico; e 3) Em operações de entrada, após a leitura de dados de periférico, o dado lido estará disponível no registrador acumulador AC, nos 8-bits menos significativos, caso o valor lido seja um byte, ou, nos 16-bits de AC, caso o valor lido seja um número inteiro. Desta forma, nas operações de E/S, o registrador acumulador AC do processador Cariri passa a incluir os campos “AC High”, utilizado para endereçamento de porta de periférico, e “AC Low”, utilizado para transferência de dados, como podem ser vistos na Figura 3.2.

Figura 3.2. Campos do registrador AC em operações de E/S.

Considerando que o periférico Video está devidamente instalado, o código a seguir ilustra os passos para impressão do caractere “A”.

Linha	Código
1	`.code`
2	`   LDA char          ;AC Low recebe 'A'`
3	`   ADD video_port    ;AC High recebe a porta de Video`
4	`   INT output        ;realiza operacao de output`
5	
6	`end:`
7	`   INT exit`
8	
9	`.data`
10	`   ;syscall exit`
11	`   exit:       DD 25`
12	`   char:       DB 'A'       ;caractere que será impresso em Video`
13	`   output:     DD 21        ;codigo de output da instrucao INT`
14	`   video_port: DD 0x0000    ;porta do periferico Video`
15	`.stack 10`

No código anterior, na Linha 12 define-se a variável “char”, que é do tipo caractere e é inicializada com o caractere “A”. Na Linha 13, é definida a variável “output”, que inclui o código de operação 21, o qual refere-se à operação da instrução INT para escrita de dados em periférico. Na Linha 14, a variável “video_port” inclui a porta do periférico Video, que por padrão tem o endereço de E/S igual a 0.

Ainda no código anterior, na Linha 2, o caractere “A” é carregado nos 8-bits menos significativos do registrador acumulador AC (campo “AC Low”). Em seguida, adiciona-se ao campo “AC High” de AC a porta do periférico Video (na Linha 3), e, por fim, na Linha 4, realiza-se a operação de escrita de dados no periférico Video, através da instrução INT com parâmetro 21 (contido na variável “output”). A Figura 3.3

apresenta, à direita, a visualização do periférico Video após a operação de escrita, onde observa-se que houve de fato a impressão do caractere “A”.

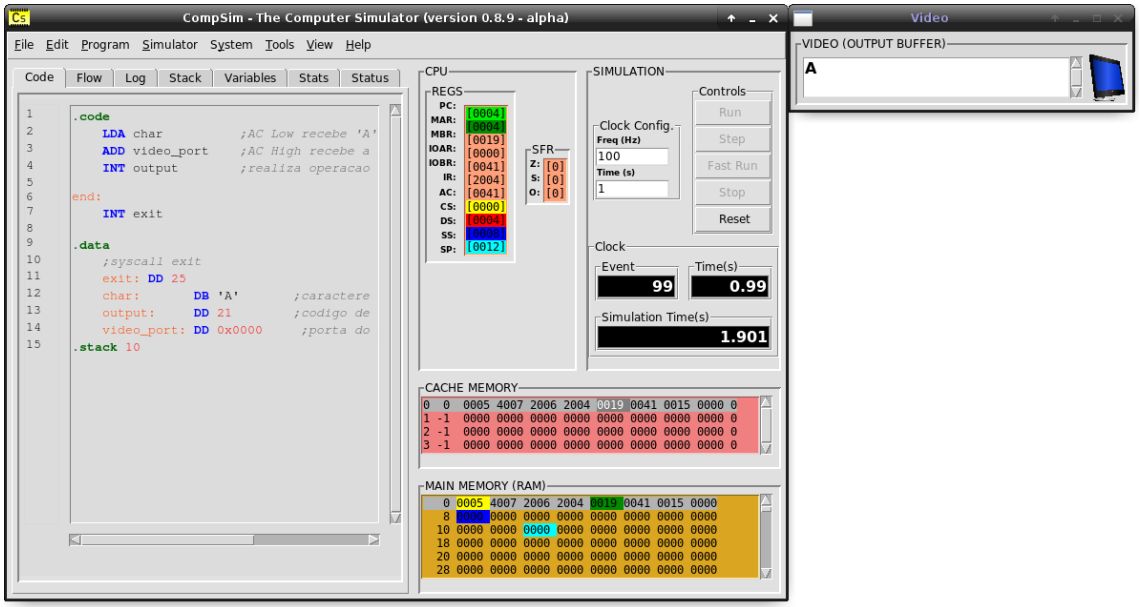


Figura 3.3. Visualização de escrita de caractere no periférico Video.

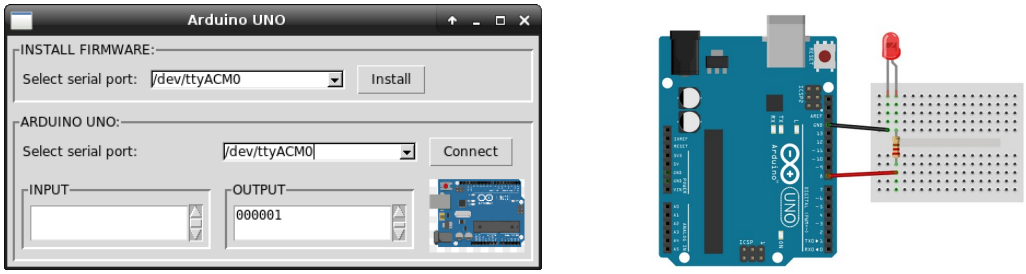
O periférico físico Arduino UNO se destaca por permitir a interação da PHV do CompSim com componentes eletrônicos reais. Arduino é uma plataforma eletrônica baseada em microcontrolador, com especificação aberta e tem sido largamente utilizada em projetos de sistemas eletrônicos [Arduino 2018].

Arduino UNO é a mais utilizada dentre as diversas versões do Arduino. Seu microcontrolador gerencia a interação com componentes eletrônicos externos através dos registradores PortB, PortC e PortD. Os bits desses registradores estão diretamente conectados aos pinos de E/S (*General Purpose Input/Output* - GPIO) em uma placa de circuitos (*board*) do Arduino UNO. Assim, através de operações de leitura e/ou escrita de dados nesses registradores é possível interagir com componentes eletrônicos conectados a uma *board* Arduino UNO, através do pinos de GPIO.

No CompSim, o periférico físico Arduino UNO, cuja interface gráfica pode ser vista na Figura 3.4(a), por padrão, mapeia os bits dos registradores PortB, PortC e PortD nas portas de 2 a 10. Com este modelo, a CPU, ao realizar operações de E/S nas portas mapeadas nos registradores de um Arduino UNO, pode ativar operações de leituras e/ou escritas digitais ou analógicas nos GPIOs.

Para ilustrar a interação entre o simulador CompSim e componentes eletrônicos reais, propõe-se o cenário exposto na Figura 3.4(b). Nele há: uma *board* Arduino UNO; uma *protoboard*, utilizada para conectar os componentes eletrônicos *led* vermelho a um resistor de 200 Ohm; e *jumpers* (fios) que conectam o circuito eletrônico montado na *protoboard* ao Arduino UNO. O *jumper* vermelho liga o GPIO 8 da *board* Arduino UNO ao resistor, que por sua vez está ligado ao polo positivo do *led*, e o *jumper* preto

liga o polo negativo do *led* ao pino de isolamento (*ground* - GND) da *board* Arduino UNO. O circuito funciona da seguinte maneira: ao realizar uma operação de escrita no registrador PortB do Arduino UNO, dependendo do valor escrito, pode-se ativar (ou desativar) uma corrente elétrica no GPIO 8, que percorrerá o circuito resistor-*led* até o GND, fazendo com que o *led* acenda.



(a) O periférico físico Arduino UNO. **(b) Circuito eletrônico para "pisca-led".**

Figura 3.4. O periférico Arduino UNO e o circuito "pisca-led".

Para a execução do cenário proposto, é necessário que: o periférico Arduino UNO esteja instalado no CompSim; o circuito apresentado na Figura 3.4(b) esteja montado e configurado; e a *board* Arduino UNO esteja conectada a uma porta USB do computador.

O passo seguinte será instalar o *firmware* no microcontrolador do Arduino UNO para que seja possível realizar a comunicação com o CompSim. Para tanto, na Figura 3.4(a), no painel "*INSTALL FIRMWARE*", deve-se escolher a porta serial na qual a *board* Arduino UNO está conectada e clicar no botão "*Install*". Em seguida, após a instalação, deve-se estabelecer uma conexão lógica entre o CompSim e a *board* Arduino UNO, selecionando novamente a porta serial e clicando no botão "*Connect*", ambos no painel "ARDUINO UNO", como mostra a Figura 3.4(a). O último passo é programar e executar uma aplicação em *Assembly* no CompSim que realize operações de escrita de dados na porta 2, a qual está mapeada no registrador PortB do microcontrolador do Arduino UNO, cujos bits estão ligados aos GPIOs 8 a 13 de uma *board*.

O código a seguir ilustra uma aplicação para ligar e desligar o led ("pisca-led") conectado na GPIO 8. No código, nas Linhas 14 e 15 estão definidos, nas variável "liga" e "desliga", os bits que serão escritos na PortB do Arduino UNO, para acender e apagar o led, respectivamente. Para a escrita dos bits em PortB, a variável "portB_arduino", definida na Linha 17, guarda a porta do periférico Arduino UNO referente ao registrador PortB. Nas Linhas 3 à 5, carrega-se os bits na variável "liga" em AC Low, acrescenta-se a porta contida na variável "portB_arduino" em AC High e realiza-se a operação de escrita para acender o *led*, respectivamente. Já as instruções nas Linhas 7 à 9 fazem com que o led desligue (os procedimentos são análogos aos de ligar o *led*, com exceção de que carrega-se em AC Low os bits da variável "desliga"). Por último, a instrução na Linha 11 realiza um desvio de fluxo de execução para o início do

programa, criando assim um loop infinito no programa e, por consequência, a aplicação ligará e desligará o *led* indefinidamente ou até que se encerre a simulação.

Linha	Código
1	`.code`
2	`loop:`
3	`   LDA liga              ;AC Low recebe os bits para ligar o led`
4	`   ADD portB_arduino     ;AC High recebe a porta de PortB`
5	`   INT output            ;realiza operacao de output`
6	
7	`   LDA desliga           ;AC Low recebe os bits para desligar o led`
8	`   ADD portB_arduino     ;AC High recebe a porta de PortB`
9	`   INT output            ;realiza operacao de output`
10	
11	`   JMP loop`
12	
13	`.data`
14	`   liga:          DD 00001b   ;5-bits para escrita em PortB`
15	`   desliga:       DD 00000b   ;5-bits para escrita em PortB`
16	`   output:        DD 21       ;codigo de output da instrucao INT`
17	`   portB_arduino: DD 0x0200   ;porta de PorB do Arduino UNO`
18	`.stack 10`

Outro periférico que se destaca é o VArduino, por apresentar um periférico virtual do Arduino UNO com disponibilidade de componentes eletrônicos simuláveis, tais como *leds*, botões, sensores, *display* de 7-segmentos, entre outros. Ele é bastante útil quando não há disponibilidade dos respectivos componentes eletrônicos físicos e apresenta o mesmo modelo de comunicação de E/S do periférico Arduino UNO (os programas escritos em *Assembly* são compatíveis entre os periféricos Arduino UNO e VArduino) [Cartaxo et al. 2020].

Finalmente, ressalta-se que o Subsistema de E/S do CompSim é bastante versátil, pois, além dos periféricos disponíveis no website do projeto, permite conectar dinamicamente, à PHV do CompSim, outros tipos desenvolvidos pelos usuários. Para tanto, na criação dos novos periféricos, para que tenham o suporte de conexão automática, deve-se implementar um modelo de interface padrão, descrito com maiores detalhes em [Esmeraldo et al. 2020].

3.4.7. Modularização de Programas

Dependendo da aplicação, os programas de computador podem se tornar maiores, por incluir mais instruções, e, por consequência, se tornarem muito difíceis de ler e compreender, realizar depuração para correções, manutenção e evolução. Ao longo dos anos, viu-se que a melhor forma de desenvolver e manter um programa maior é modularizá-lo, através da sua divisão em partes menores (também chamadas de "módulos"), que são mais simples de tratar em relação ao programa como um todo. Há vários motivos para modularizar um programa, entre eles estão: 1) Dividir um problema maior em problemas menores, que são mais simples de tratar, e agrupar suas soluções para compor a solução final do problema maior; 2) Evitar repetição de código, onde ao se criar funções, que são blocos de instruções que realizam tarefas específicas, elas

podem ser chamadas para execução sob demanda, em diferentes pontos do programa; e 3) Abstrair o código, pois ao chamar uma função para execução, não é necessário compreender como a mesma está implementada, apenas utiliza-se a sua assinatura, como são os casos da funções “printf”, “scanf” e “malloc”, da linguagem de programação C.

A CPU do CompSim inclui duas instruções que implementam, respectivamente, os passos para chamada (CALL) e retorno (RET) de uma função. No CompSim, para a definição de uma função basta definir um bloco de instruções, em que a primeira delas terá um rótulo, que será o identificador da função, e a última será a instrução RET. Com isso, pode-se utilizar a instrução CALL com o identificador da função para chamá-la para execução (o endereço da instrução de retorno é adicionado na pilha do programa e o fluxo de execução do programa é desviado para a primeira instrução do bloco de instruções da função). Ao final da execução da função, a instrução RET faz com que o fluxo de execução seja novamente desviado, retornando à instrução posterior à da chamadora da função, cujo endereço foi previamente guardado na pilha do programa.

O código a seguir ilustra a criação de uma função que realiza a impressão do caractere “A” no periférico Video e como ela pode ser chamada para execução.

Linha	Código
1	`.code`
2	`   CALL printA`
3	`   CALL printA`
4	`   CALL printA`
5	
6	`end:`
7	`   INT exit`
8	
9	`printA:`
10	`   LDA char          ;AC Low recebe 'A'`
11	`   ADD video_port    ;AC High recebe a porta de Video`
12	`   INT output        ;realiza operacao de output`
13	`   RET`
14	
15	`.data`
16	`   ;syscall exit`
17	`   exit:       DD 25`
18	`   char:       DB 'A'       ;caractere que sera impresso em Video`
19	`   output:     DD 21        ;codigo de output da instrucao INT`
20	`   video_port: DD 0x0000    ;porta do periferico Video`
21	`.stack 10`

No código anterior, nas Linhas 9 à 13, encontra-se a definição da função chamada “printA”, em que: na Linha 9, observa-se o rótulo “printA” da instrução na Linha 10 (o rótulo consiste do identificador da função); o bloco de instruções nas Linhas 10 à 12 realiza a impressão do caractere “A” no periférico Video; e, na Linha 13, a instrução RET encerra a execução da função, ao realizar o desvio de fluxo de execução para retorno. Observa-se, nas Linhas 1 à 3, que a função “printA” é chamada 3 vezes para execução, por meio da instrução “CALL printA”. Ao executar esse

programa, espera-se que sejam impressos 3 caracteres “A” no periférico Video, como mostra a Figura 3.5.

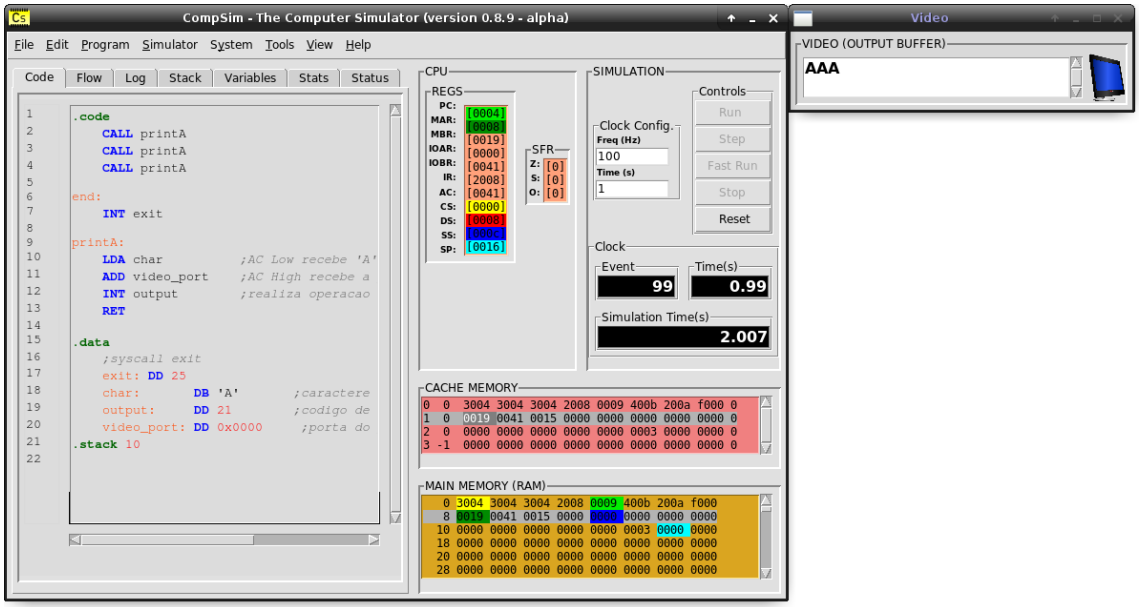


Figura 3.5. Visualização de escrita de caracteres no periférico Video por chamadas de função.

O código a seguir ilustra a modularização do código para ligar e desligar o *led* conectado na GPIO 8 (programa “pisca-*led*”).

Linha	Código
1	`.code`
2	`loop:`
3	`    CALL ligaLed`
4	`    CALL desligaLed`
5	`    JMP loop`
6	
7	`ligaLed:`
8	`    LDA liga              ;AC Low recebe os bits para ligar o led`
9	`    ADD portB_arduino     ;AC High recebe a porta de PortB`
10	`    INT output            ;realiza operacao de output`
11	`    RET`
12	`desligaLed:`
13	`    LDA desliga           ;AC Low recebe os bits para desligar o led`
14	`    ADD portB_arduino     ;AC High recebe a porta de PortB`
15	`    INT output            ;realiza operacao de output`
16	`    RET`
17	
18	`.data`
19	`    liga:          DD 00001b   ;5-bits para escrita em PortB`
20	`    desliga:       DD 00000b   ;5-bits para escrita em PortB`
21	`    output:        DD 21       ;codigo de output da instrucao INT`
22	`    portB_arduino: DD 0x0200   ;porta de PorB do Arduino UNO`
23	`.stack 10`

Nas Linhas 7 à 11, está definida a função “ligaLed”, cujas instruções fazem com que seja ativada a corrente elétrica que liga o led, e, nas Linhas 12 à 16, define-se a função “desligaLed”, responsável por desativar a corrente elétrica. As Linhas 2 à 5 implementam um *loop* infinito, em que a instrução da Linha 3 chama a função “ligaLed” e a instrução da Linha 4 chama a função “desligaLed”. Dessa forma, a aplicação executará com a chamada iterativa das funções para ligar e desligar o *led* indefinidamente ou até que se encerre a simulação.

Com os recursos do simulador CompSim, além da definição e chamada de funções para execução, é possível explorar outros aspectos, tais como: passagem de parâmetros por valor e por referência; passagem de parâmetros em variáveis globais, registradores e pilha do programa; retorno de função; implementação de variáveis locais, através de variáveis globais, alocação de memória e pilha do programa; definição e execução de funções aninhadas e de funções recursivas.

3.5. Conclusões

Este capítulo apresentou uma proposta de abordagem de uso do simulador CompSim para apoio ao aprendizado prático de conceitos de Arquitetura e Organização de Computadores. O simulador abordado inclui uma plataforma de hardware virtual que contém os principais componentes do computador, tais como CPU, memórias, barramentos e periféricos, cujas características são semelhantes às dos respectivos componentes reais. Além disso, o simulador conta com uma interface gráfica, que inclui recursos integrados para simplificar a configuração, programação, simulação, visualização e análise de desempenho de sistemas computacionais criados no ambiente virtual.

Ao longo do capítulo, foram apresentados alguns dos recursos do simulador CompSim e como eles podem ser utilizados para estimular a aplicação prática dos conceitos aprendidos em aulas teóricas e, com isso, otimizar o processo de ensino-aprendizagem em projetos de sistemas computacionais. Observa-se que os elementos apresentados neste capítulo não esgotam o leque de recursos e potencialidades de aprendizagem pelo uso do simulador CompSim. Desta forma, recomenda-se explorar o website do projeto [CompSim 2021], onde estão disponíveis, além do simulador, materiais didáticos variados, relação de publicações científicas, link para grupo de discussão sobre o projeto, entre outros. Sugere-se ainda a leitura dos trabalhos: [Lisboa et al. 2018], que detalha a interface de comunicação entre o CompSim-Arduino UNO; [Esmeraldo et al. 2018], que traz descrições de mais recursos gráficos do simulador; [Cartaxo et al. 2020], para aprofundamento teórico na dinâmica de simulação de sistemas computacionais com o periférico VArduno; e [Esmeraldo et al. 2020], para compreender a interface do subsistema de E/S e como pode-se criar novos periféricos virtuais e físicos para a PHV do CompSim.

Cabe destacar que o CompSim tem sido utilizado por diferentes turmas de cursos de Técnico Integrado em Eletrônica e de Bacharelado em Sistemas de Informação e os resultados vêm mostrando que, ao utilizar o simulador CompSim em

aulas práticas, o aprendizado em projetos de sistemas computacionais se torna mais atrativo, dinâmico, produtivo e efetivo.

Por fim, ressalta-se que o simulador está em contínuo desenvolvimento e que o feedback dos estudantes e o apoio de diversas instituições parceiras têm sido muito importantes para adição de novos recursos, bem como para o aprimoramento do simulador e da experiência de uso, bem como do aprendizado em projetos de sistemas computacionais.

Referências

[ACM and IEEE 2013] ACM, Association for Computing Machinery and IEEE Computer Society (2013) "Curriculum Guidelines for Undergraduate Degree Programs in Computer Science", The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society.

[Arduino 2018] Arduino (2018) "What is Arduino?" https://www.arduino.cc/en/Guide/Introduction

[Bahk et al. 2013] Bahk, J. H.; Youngs, M.; Yazawa, K.; Shakouri, A. and Pantchenko, O. (2013) "An online simulator for thermoelectric cooling and power generation", In: Frontiers in education Conference, IEEE, 2013.

[Balamuralithara and Woods 2009] Balamuralithara, B. and Woods, P. C. (2009) "Virtual laboratories in engineering education: The simulation lab and remote lab", In: Computer Applications in Engineering Education, Vol. 17(1). pp. 108–118.

[Cartaxo et al. 2020] Cartaxo, L. F., Mendes, C. S. R., Lisboa, E. B. and Esmeraldo, G. A. R. M. (2020) "Utilizando VArduino para Criação de Periféricos Virtuais baseados em Arduino UNO para o Simulador CompSim", In: Revista Tecnologias na Educação, v. 33, pp. 1-17.

[CompSim 2021] CompSim (2021) "CompSim - The Computer Simulator". http://compsim.crato.ifce.edu.br/

[Esmeraldo et al. 2018] Esmeraldo, G., Cartaxo, L. F., Mendes, C. S. R. and Lisboa, E. B. (2018) "Um Simulador Educacional para Apoio ao Projeto de Sistemas Computacionais: Hardware, Software e suas Interfaces", In: XXVI Workshop sobre Educação em Computação (WEI 2018) - XXXVIII Congresso da Sociedade Brasileira de Computação (CSBC 2018).

[Esmeraldo et al. 2019] Esmeraldo, G. A. R. M., Mendes, C. S. R., Cartaxo, L. F. and Lisboa, E. B. (2019) "Apoio ao Aprendizado em Arquitetura e Organização de Computadores: Um Estudo Comparativo entre Simuladores Computacionais", In: Revista Tecnologias na Educação, v. 31, pp. 1-17.

[Esmeraldo et al. 2020] Esmeraldo, G. A. R. M., Lisboa, E. B., Mendes, C. S. R., Cartaxo, L. F., Ribeiro, C. V., Morato, L. F. B., Santos, P. S. and Nascimento, M. S. (2020) "Uma Abordagem Integrada de Hardware e Software para o Aprendizado de Subsistemas de Entrada/Saída em Projetos de Sistemas Computacionais", In: International Journal of Computer Architecture Education, v. 9, pp. 1-9.

[Garcia, Pacheco and Garcia 2014] Garcia, I. A.; Pacheco, C. L. and Garcia, J. N. (2014) “Enhancing education in electronic sciences using virtual laboratories developed with effective practices”, In: Computer Applications in Engineering Education, Vol. 22(2), pp. 283–296.

[Keutzer et al. 2000] Keutzer, K., Newton, A. R., Rabaey, J. M. and Sangiovanni-Vincentelli, A. (2000) "System-level design: orthogonalization of concerns and platform-based design", In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19(12), pp. 1523-1543.

[Lisboa et al. 2018] Lisboa, E. B., Cartaxo, L. F., Mendes, C. S. R. and Esmeraldo, G. A. R. M. (2018) “Ambiente Integrado de Hardware e Software Aplicado ao Ensino de Projeto de Sistemas Computacionais”, In: III Congresso sobre Tecnologias na Educação (Ctrl+E 2018).

[Lisboa et al. 2019] Lisboa, E. B., Cartaxo, L. F., Mendes, C. S. R. and Esmeraldo, G. A. R. M. (2019) “Uma Metodologia Educacional para Aprendizado Prático de Organização e Arquitetura de Computadores com Apoio de Simulador Computacional”, In: Brazilian Journal of Development, v. 5, pp. 31062-31068.

[Monteiro 2007] Monteiro, M. A. (2007) “Introdução à Organização de Computadores”. 4a. Ed. LTC.

[Nikolic et al. 2009] Nikolic, B.; Radivojevic, Z.; Djordjevic, J. and Milutinovic, V. A. (2009) “Survey and Evaluation of Simulators Suitable for Teaching Courses in Computer Architecture and Organization”, In: IEEE Transactions on Education, Vol. 52, No. 4.

[Null and Lobur 2009] Null, L. and Lobur, J. (2009) “Princípios Básicos de Arquitetura e Organização de Computadores”. Ed. Bookman.

[Patrick, Fardo and Chandra, 2020] Patrick, D. R., Fardo, S. W. and Chandra, V. (2020) “Electronic Digital System Fundamentals”. 1st Ed. River Publishers.

[SBC 2005] SBC. Sociedade Brasileira de Computação (2005) “Currículo de Referência da SBC para Cursos de Graduação em Bacharelado em Ciência da Computação e Engenharia de Computação”. https://www.sbc.org.br/documentos-da-sbc/category/131-curriculos-de-referencia

[Santos and Langlois 2018] Santos, P. R. and Langlois, T. (2018) “Compiladores: da Teoria à Prática”. LTC.

[Stallings 2010] Stallings, W. (2010) “Computer Organization and Architecture. Designing for Performance”. 8th Ed. Prentice Hall.

[Tanenbaum and Austin 2013] Tanenbaum, A. S. and Austin, T. (2013) “Organização estruturada de computadores”, 6a. Ed. Pearson.

[Uribe et al. 2016] Uribe, M. D. R.; Magana, A. J.; Bahk, J.-H. and Shakouri, A. (2016) “Computational Simulations as Virtual Laboratories for Online Engineering Education: A Case Study in the Field of Thermoelectricity”, In: Computer Applications in Engineering Education, Vol. 24(3). pp. 428–442.

[Wolffe et al. 2002] Wolffe, G. S.; Yurcik, W.; Osborne, H.; Holliday and M. A. (2002) "Teaching computer organization/architecture with limited resources using simulators", In: Proceedings of the 33rd SIGCSE technical symposium on Computer science education, ACM SIGCSE Bulletin. Vol. 34, No. 1.