INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA PARAÍBA Campus Campina Grande	**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA – CAMPUS CAMPINA GRANDE**		
	CURSO:	**CURSO SUPERIOR ENGENHARIA DA COMPUTAÇÃO**	
	PERÍODO:		**TURMA:**
	DISCIPLINA:	**ALGORITMOS E LABORATÓRIO DE PROGRAMAÇÃO**	
	PROFESSOR:	**CÉSAR ROCHA VASCONCELOS**	SEMESTRE LETIVO

Módulo 01 – ALGORITMOS & CONCEITOS FUNDAMENTAIS

1. O COMPUTADOR

Existem várias definições para o que seja um computador. Podemos dizer que, de maneira geral, um computador é uma coleção de componentes físicos (placas, cabos, etc.) e lógicos (programas) capaz de receber, processar, transformar, armazenar e retornar informações aos seus usuários. Contudo, para que o computador possa realizar alguma tarefa ou resolver algum problema (por exemplo: calcular a média de um aluno, listar os clientes de uma locadora de vídeo, etc.), é necessário que ele saiba, com precisão, qual seqüência de passos adequada deve ser executada para atingir estes objetivos.

1.1 Organização de um computador

Para facilitar nosso entendimento e não ficarmos "amarrados" a um modelo específico de computador, descreveremos uma máquina hipotética. É ela que será usada em nossos estudos para validar nossos experimentos algorítmicos. Observe, na Figura 1 abaixo, sua organização:

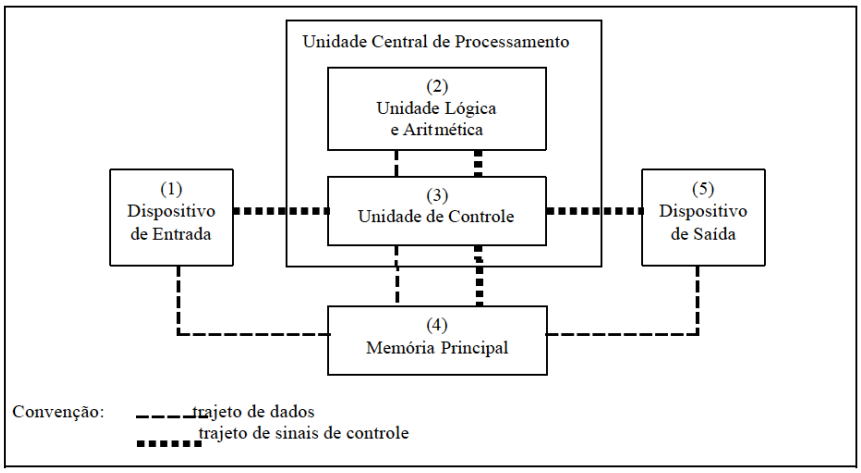

Figura 1: Uma máquina hipotética para escrever nossos algoritmos. [1]

Todos os computadores, independentemente dos seus tamanhos, modelos e marcas, são conceitualmente semelhantes ao esquema da figura anterior (há algumas diferenças, mas não trataremos aqui dos casos especiais).

Cada uma das partes de nossa máquina hipotética pode ser definida da seguinte forma [1]:

a) **Dispositivo de entrada** (o teclado): será o dispositivo de hardware pelo qual os dados que serão trabalhados pelo algoritmo vão ser introduzidos em nosso computador hipotético;
b) **Unidade Lógica e Aritmética** (ULA): parte responsável pelas operações matemáticas e avaliações lógicas;
c) **Unidade de Controle** (UC): esta unidade exerce controle sobre as demais partes do nosso computador. É uma verdadeira gerente que coordena e distribui tarefas às outras unidades; é ela, também, quem busca uma instrução na memória e a decodifica. Dependendo do tipo da

instrução, pode haver uma transferência do controle para a ULA ou para os componentes externos à CPU;

d) **Memória principal (RAM)**: guarda as instruções a serem executadas e os dados a serem utilizados pelas mesmas. Todo dado fornecido ao computador e o resultado de suas operações (processamento realizado) ficam guardados na memória;

e) **Dispositivo de Saída** (vídeo e impressora): é o meio que se dispõe para apresentação dos resultados obtidos.

Importante: a **Unidade Central de Processamento,** também conhecida como CPU (*Central Processing Unit*), é responsável pela execução dos programas e pelo comportamento (coordenação) das outras unidades no sistema. É capaz de fazer contas matemáticas e fazer decisões simples. As principais partes da CPU são: a Unidade Lógica e Aritmética, Unidade de Controle e Memórias (Registradores, Memória ROM e Cache)[1].

2. DEFINIÇÃO DE ALGORITMO

Na seção anterior, foi discutida a organização interna de uma máquina hipotética. Esta máquina irá nos ajudar a entender melhor como o computador realiza o processamento de instruções para uma tarefa qualquer. Nesta seção, entretanto, iremos começar a escrever os nossos primeiros algoritmos.

De maneira simples, podemos definir um algoritmo como sendo:

> ***"Uma seqüência ordenada, finita e sem ambigüidade de passos que leva à solução de um determinado problema".***

2.1 Exemplos de algoritmos no nosso cotidiano:

- a seqüência de passos para realizar uma ligação a cobrar presente nos orelhões;
- uma receita de um bolo;
- o manual de instalação/utilização presente em alguns equipamentos eletrônicos;
- a maneira como uma pessoa toma banho [1];
- uma estratégia definida por um competidor para vencer um jogo;
- os passos para trocar o pneu de um carro;
- etc.

2.2 Por que é importante saber escrever bons algoritmos?

Uma resposta curta seria: *"porque você irá definir uma seqüência de passos para que um computador a execute e, assim, realize uma tarefa para você"*. Dependendo de quão boa (ou ruim) esta seqüência de passos é montada, o objetivo pretendido pode ser atingido com sucesso ou não. Além disso, isso é o que normalmente diferencia um bom de um mau programador.

De acordo com [1], a importância do algoritmo está no fato de termos que especificar uma seqüência de passos lógicos para que o computador possa executar uma tarefa qualquer, pois o mesmo por si só não tem vontade própria, faz apenas o que mandamos. Com uma ferramenta algorítmica, podemos conceber uma solução para um dado problema, independendo de uma linguagem específica em que o algoritmo foi escrito e até mesmo do próprio computador.

Quando queremos que o computador realize uma tarefa para nós, devemos programá-lo. De maneira geral, para se programar um computador com sucesso, três fatores devem ser considerados:

[1] Procure informações em livros ou mesmo na Internet sobre estes componentes que não aparecem na figura: Memória ROM, Memória Cache e os chamados Registradores. Verifique a função e a importância de cada um para o computador.

(a) identificar e entender bem o problema que se está querendo resolver[2]; (b) escrever uma solução algorítmica e, finalmente, (c) escrever (mapear) este algoritmo em uma linguagem de programação real[3].

2.3 Características de um algoritmo:

- os passos que compõem o algoritmo devem ser simples, claros e sem ambigüidade;
- os passos devem estar numa ordem (numa seqüência) cuidadosamente definida;
- o algoritmo deve ser efetivo, isto é, deve resolver o problema num número finito de passos.
- o algoritmo deve tratar todas situações não previstas (eventos extraordinários)

Exemplo: um algoritmo para trocar uma lâmpada.

1ª Tentativa:

1. Remova a lâmpada queimada;
2. Coloque a lâmpada nova;

Nesta primeira versão, usamos o conceito de **sequenciação**. Todo algoritmo é uma seqüência de passos. A sequenciação é aplicada quando a solução do problema pode ser decomposta em passos individuais. Os comandos do algoritmo fazem parte de uma seqüência, onde é relevante a ordem na qual se encontram os mesmos. Cada um dos comandos será executado um de cada vez, estritamente, de acordo com essa ordem definida.

Entretanto, o que você achou deste algoritmo? Bom? Ruim? Verifique se ele atende a todas as características discutidas há pouco. Todas as características são atendidas? Não! Este algoritmo, apesar de parecer claro, na verdade, está muito genérico.

2ª Tentativa

1. Remova a lâmpada queimada;
   1.1. Posicione a escada debaixo da lâmpada;
   1.2. Suba na escada até que a lâmpada possa ser alcançada;
   1.3. Gire a lâmpada no sentido anti-horário até que solte;
2. Coloque a lâmpada nova;
   2.1. Escolha uma lâmpada nova da mesma potência que a queimada;
   2.2. Posicione a nova lâmpada no soquete;
   2.3. Gire-a no sentido horário até que se firme;
   2.4. Desça da escada;

Esta segunda versão está melhor. Os passos 1 e 2 foram desdobrados (**refinados**) em passos detalhados de maneira a torná-los mais claros e precisos. A este desdobramento, dá-se o nome de **refinamento sucessivo**. Muitos programadores (ainda hoje), diante de um problema algorítmico, tendem a escrever primeiro os passos gerais (o arcabouço do algoritmo) e, só depois, aplicar um desdobramento dos passos até que se atinja o resultado desejado.

Poderíamos refinar o passo 2.1 deste algoritmo ainda mais. Quer ver?

---

[2] Acredite, este é um dos principais problemas encontrados pelos profissionais de informática quando se deseja construir um sistema de software para um cliente. Isso se chama **Levantamento de Requisitos**.

[3] Linguagem que independe do conjunto de instruções da linguagem de máquina de um computador. Cada instrução de alto nível equivale a várias instruções da linguagem de máquina, sendo, portanto, mais produtiva, legível e amigável a nós humanos. Ex.: Pascal, C, Java, Algol, BASIC, Lisp, Prolog, etc.

3ª Tentativa

1. Remova a lâmpada queimada;
   1.1. Posicione a escada debaixo da lâmpada;
   1.2. Suba na escada até que a lâmpada possa ser alcançada;
   1.3. Gire a lâmpada no sentido anti-horário até que solte;
2. Coloque a lâmpada nova;
   2.1. Escolha uma lâmpada nova da mesma potência que a queimada;
      2.1.1. Selecione uma nova lâmpada na caixa de lâmpadas
      2.1.2. Verifique a potência
      2.1.3. Se for igual a da lâmpada queimada então pare o processo de busca na caixa
      2.1.4. Senão, repita a operação a partir do passo 2.1.1
   2.2. Posicione a nova lâmpada no soquete;
   2.3. Gire-a no sentido horário até que se firme;
   2.4. Desça da escada;

Esta terceira versão utilizou ainda mais o conceito de **refinamento sucessivo**. Mais uma vez, note que o que fizemos foi apenas desdobrar o passo 2.1 em instruções cada vez menores e mais claras. Entretanto, neste novo desdobramento, utilizamos dois novos conceitos importantes: **decisão**[4] (veja o teste sublinhado usando SE/SENÃO no algoritmo para ver se achamos uma lâmpada de mesma potência) e **iteração** (veja a operação pontilhada usando REPITA no algoritmo, instruindo o computador a repetir um bloco de comandos quando a potência da lâmpada escolhida é diferente do que se deseja).

Você deve estar se perguntando: **Como saber se o algoritmo está detalhado o suficiente?**

De acordo com [2], a resposta é: **depende de quem vai executá-lo**. Não devemos definir os passos de forma muito genérica (ou abstrata), nem tampouco detalhar demasiadamente os passos a ponto de tornar o algoritmo óbvio e grande demais.

O mesmo autor nos cita ainda que, para um computador, o algoritmo deve ser escrito de tal forma que cada passo se aproxime o máximo de suas instruções básicas de alto nível (em outras palavras, comandos de uma linguagem de programação). À medida que se adquire prática, vai-se abstraindo os detalhes básicos do algoritmo. O ideal é um algoritmo que não seja muito próximo de um programa (senão seria melhor escrever de uma vez o programa) nem muito superficial. Apenas a prática na escrita de algoritmos e posteriormente a elaboração de programas é que faz o programador encontrar este meio termo ideal.

Questões: escreva os seguintes algoritmos (Dica: comece com passos genéricos e utilize refinamentos sucessivos, decisões e iterações, quando desejar).

- Algoritmo com instruções para trocar o pneu de um carro usando uma chave de roda, um macaco e um estepe (em boas condições).
- Algoritmo para fritar um ovo pela manhã usando uma panela, sal, manteiga e um garfo;
- Algoritmo para fazer uma chamada telefônica local; inclua as possibilidades de a chamada local ser a cobrar ou simples;

3. FORMAS DE REPRESENTAÇÃO DE UM ALGORITMO

Na seção anterior, vimos que um algoritmo é um dos primeiros passos na preparação de um programa de computador. Além disso, a definição de algoritmos bem como suas características principais foram apresentadas e discutidas. Ademais, foi mostrado um exemplo de um algoritmo expresso em uma linguagem natural (no caso, uma seqüência simples de passos descritos em português).

[4] Ou também conhecida por **Seleção.**

Existem, na verdade, diversas formas de se representar um algoritmo. Segundo [1], os algoritmos podem ser expressos das seguintes formas:

3.1 Narrativa

Na verdade, são apenas algoritmos expressos em **linguagem natural**. Esta abordagem é pouco formal, não padronizada, mas ótima quando se deseja efetuar rascunhos iniciais (*brainstorming*) apenas para ter uma idéia geral do algoritmo e seus passos gerais (todo o seu esqueleto).

Exemplo: Receita de um bolo.

1ª Tentativa:

1. Providencie manteiga, ovos, 2 Kg de massa, etc.
2. Misture todos os ingredientes numa tigela
3. Despeje a mistura na fôrma de bolo
4. Ligue o forno e ajuste-o em 220º C
5. Leve a fôrma ao forno
6. Espere 20 minutos
7. Retire a fôrma do forno
8. Deixe esfriar
9. Prove

A partir deste esqueleto inicial, poder-se-ia fazer refinamentos sucessivos em diversos passos.

3.2 Fluxograma

Ao contrário dos algoritmos expressos em linguagem natural, esta abordagem, por outro lado, é padronizada. Existem símbolos padronizados para início, entrada de dados, cálculos, saída de dados, fim, etc. Para exemplificar, suponha um algoritmo chamado de "Cálculo do Dobro de um Número". Este algoritmo apenas recebe um número inicial qualquer, calcula o seu dobro e retorna o resultado desta operação.

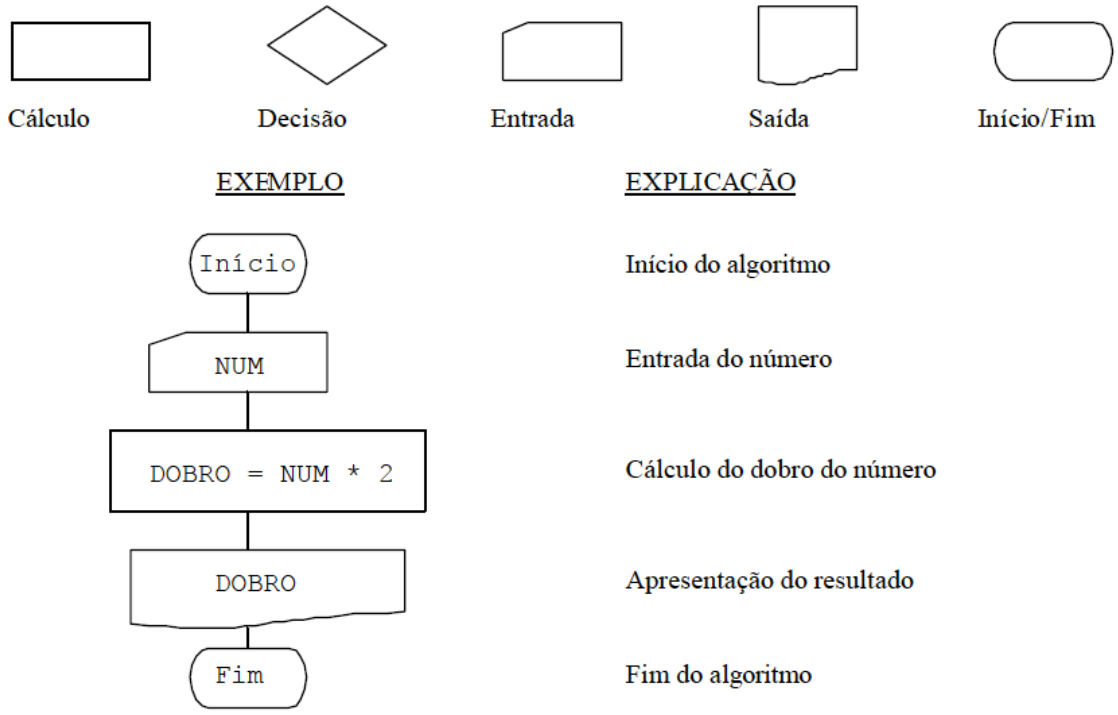

Figura 2: Algoritmo "Calcula Dobro de um Número" expresso em fluxograma [1]

3.3 Linguagem algorítmica

De acordo com [1], são algoritmos expressos em uma pseudolinguagem de programação. Na verdade, são representações que misturam um pouco de uma linguagem de programação real com um pouco da abordagem narrativa. Apesar de não ser baseada em nenhuma linguagem de programação real, pode ser facilmente migrada para uma. Também, faz-se necessário definir que tipos de dados serão representados, bem como alguns comandos básicos de entrada e saída. Esta será a notação mais usada em nossos primeiros algoritmos.

EXEMPLO:

```
Algoritmo CALCULA_DOBRO
   NUM,DOBRO : inteiro
início
   Leia NUM
   DOBRO ← 2 * NUM
   Escreva DOBRO
fim
```

Figura 3: Algoritmo "Calcula Dobro de um Número" expresso em uma pseudolinguagem [1]

4. CICLO DE EXECUÇÃO DE UM ALGORITMO EM NOSSO COMPUTADOR HIPOTÉTICO

Na seção 1.1, Organização de um computador, definimos um computador genérico capaz de executar um conjunto de instruções. Para não somente validar nosso algoritmo "Cálculo do Dobro de um Número", visto há pouco, mas, sobretudo, observar a interação entre as partes internas da máquina hipotética durante a execução deste programa, vamos submeter este conjunto de instruções ao nosso computador hipotético. Observe que a CPU executaria o algoritmo da seguinte forma:

1. A UC (Unidade de controle) pede que a primeira instrução do algoritmo seja enviada à uma memória interna do processador (registrador de instrução).
2. A UC decodifica a instrução como sendo uma instrução de entrada (o pedido do número a ser usado como base para o cálculo do dobro). Ela, então, envia a ordem e um sinal de controle para um dispositivo de saída do computador para que o número seja pedido ao usuário (uma mensagem no monitor: "Insira o número: ", por exemplo).
3. O usuário digita o número. A UC, então, recebe um sinal do dispositivo de entrada que o número foi inserido pelo teclado e já se encontra disponível. A UC, a partir daí, avisa a memória principal que receba e que guarde este número.
4. A UC pede que a segunda instrução do algoritmo seja enviada à uma memória interna do processador (registrador de instrução)
5. Decodificada a instrução (agora com a instrução calcular o dobro), a UC sinaliza a memória principal que envie o número digitado para um registrador interno da CPU (registrador de dados).
6. De posse do número, a UC pede que a ULA calcule o dobro deste número. A ULA, então calcula o dobro e coloca o resultado em um registrador interno da CPU (registrador de dados).
7. A UC, finalmente, pede que a última instrução do algoritmo seja enviada à uma memória interna do processador (registrador de instrução)
8. Decodificada a instrução (apresente o resultado), a UC envia o resultado do registrador de dados para um dispositivo de saída (monitor, por exemplo).

Em resumo, podemos afirmar que existem 4 (quatro) operações básicas que qualquer computador pode executar:

1. **operações de entrada e saída:** ler dados do teclado e escrever dados na tela são exemplos destas operações. Elas servem para introduzir dados na memória do nosso computador e exibir dados que já estejam lá armazenados;
2. **operações aritméticas:** são utilizadas na realização de operações matemáticas (adição, subtração, multiplicação e divisão);
3. **operações lógicas e relacionais:** têm aplicabilidade em comparações, testes de condições lógicas (2 > 6 ? ou ainda, X=Y ?);
4. **movimentação de dados entre os vários componentes:** as operações aritméticas são executadas na Unidade Lógica e Aritmética, necessitando da transferência dos dados para essa unidade e da volta do resultado final para ser guardado na memória.

5. LINGUAGENS DE PROGRAMAÇÃO DE ALTO NÍVEL

Apesar de se poder implementar um algoritmo em uma pseudolinguagem ou mesmo em uma linguagem de programação real, é importante lembrar que as instruções definidas em um algoritmo precisam ser traduzidas para uma linguagem de máquina-alvo. Conforme dito no início desta nota de aula, o computador é composto de um conjunto de componentes físicos e eletrônicos. Para operar estes componentes diretamente, é necessário conhecer o conjunto de instruções de baixo nível aceito por cada componente específico. A este conjunto de instruções que pode ser interpretado e executado diretamente pela CPU de um dado computador dá-se o nome de **linguagem de máquina.**

5.1 Linguagem de máquina:

Em poucas palavras, um programa em linguagem de máquina é uma seqüência de conjuntos de bits que representam instruções que podem ser executaras diretamente pelo computador (ex: 00100010 = Somar ou 10100011 = Subtrair). Assim, você já deve ter percebido que escrever softwares em linguagem de máquina é uma tarefa muito complexa e cansativa; os seres humanos usam uma linguagem com nível muito mais elevado, chamada de **linguagem natural**.

Como muitos computadores possuem características diferentes, é necessário, portanto, uma maneira de traduzir estas instruções dos usuários em linguagem de alto nível para uma linguagem de máquina específica de computador-alvo. Afinal, programar diretamente em linguagem de máquina dificulta o entendimento e compreensão do código gerado. Isso porque:

- não estamos acostumados com a notação binária (seqüências de números 0 e 1 não significam absolutamente nada para nós).
- escrever um programa em binário é cansativo!
- qualquer programa, ainda que pequeno, necessita de muitas instruções para realizar a tarefa desejada.
- a possibilidade do programador cometer erros é grande (a simples inversão de um número 0 com um número 1 ou vice-versa, pode trazer conseqüências imprevisíveis. E isso é o suficiente para que o computador compreenda outra instrução totalmente diferente da que se desejava originalmente.
- além disso, a localização de erros dentro do programa é muito difícil. Afinal, todo o programa em si é uma enorme cadeia formada de zeros e uns.

Portanto, a linguagem de máquina é de difícil aprendizado e pouco expressiva para as pessoas. Se fôssemos programar diretamente em linguagem de máquina, precisaríamos lembrar cada uma das seqüências possíveis (ou significados) de cada uma das instruções que uma máquina pode realizar.
Assim, para tornar a atividade de programação mais fácil e acessível, foram desenvolvidas outras linguagens denominadas de **linguagens de programação**, que funcionam (até hoje) como uma forma alternativa interessante de se comunicar com o computador e instruí-lo a realizar algo para nós.

As linguagens de programação são compostas por um grupo de elementos e regras que permitem a construção das instruções utilizadas para resolver os problemas computacionais. Com elas, construímos programas que devem ser, posteriormente, transformados em instruções em linguagem de máquina. Para realizar a transformação, cada linguagem de programação possui um programa-suporte denominado, genericamente, de **tradutor**.

Em função das dificuldades apresentadas pela linguagem de máquina, procurou-se soluções para facilitar o trabalho do programador:

- substituir as seqüências de 0 e 1 por **mnemônicos (**isto é, abreviações que facilitam a memorização e a compreensão por parte do programador);
- toda a manipulação de conteúdos de memória passa a ser realizada por lugares especialmente reservados para este fim: os registradores (comentados anteriormente).

Logo, ao invés de:

- 10010001 (Carregue o conteúdo da primeira memória)
- 10010010 (Carregue o conteúdo da segunda memória)
- 01010010 (Realize a soma das duas memórias)
- 10001000 (Transfira o resultado para uma terceira)

Teríamos:

- LOAD A , 8 (Carregue o conteúdo do registrador A com o valor 8)
- LOAD B , 2 (Carregue o conteúdo do registrador B com o valor 2)
- ADD A , B (Realize a soma dos registradores e armazena em A)
- LOAD C , A (Transfira o resultado para um terceiro registrador)

Neste caso, já se pode verificar uma maior facilidade para compreender qual a tarefa está sendo realizada. Observe, também, que as seqüências de números foram substituídas por abreviações (como LOAD, que significa movimentar ou carregar ou ainda ADD, que significa somar, adicionar). Este conjunto de abreviações ou mnemônicos recebeu o nome de linguagem *Assembly* (do inglês, MONTAGEM).

Por sua vez, o processo de transformação destes mnemônicos em código de máquina recebeu o nome de *Assembler* (do inglês "montador"). Muitos afirmam que este processo não é exatamente uma tradução - é simplesmente uma substituição dos símbolos que compreendemos com maior facilidade nas respectivas cadeias de 0's e 1's a serem executadas pelo computador, ou seja, é a transformação da abreviação ADD em 01010010. Por este motivo, na linguagem *Assembly*, não ocorrem os processos de Compilação ou Interpretação (a serem comentados mais à frente), pois as instruções já representam ordens para a máquina diretamente.

Cada um dos mnemônicos está ligado diretamente a uma instrução do processador. Assim, o *Assembly* é dependente da arquitetura interna do processador. O *Assembly* é, então, uma linguagem de baixo nível.

5.2 Linguagens de programação de alto nível

Embora representem uma grande evolução em relação à linguagem de máquina, as linguagens de montagem ainda estavam muito distantes da linguagem natural. Isso dificultava o trabalho do programador! Surgiram, então, as chamadas **linguagens de programação de alto nível**.

Pascal, Java, C, COBOL e FORTRAN são exemplos de linguagens de programação reais de alto nível. São chamadas assim, pois têm uma estrutura de representação dos comandos mais adequada ao entendimento humano e não da máquina.

Exemplo: um algoritmo em uma linguagem de programação real (Java) para calcular a média de um aluno com base em duas notas já conhecidas:

```
public class CalcularMédia {

	public static void main(String[] args) {
		// as notas do aluno
		double nota1 = 7.5;
		double nota2 = 8.5;

		// o cálculo da média
		double média = (nota1 + nota2) / 2;

		// estrutura de decisão (aprovado ou reprovado?)
		if ( média >= 7 ) {
			System.out.println("Aprovado!");
		}else {
			System.out.println("Reprovado!");
		}

		// sair do sistema
		System.exit(0);
	}
}
```

6. COMPILADORES E INTERPRETADORES:

Viu-se, anteriormente, alguns conceitos fundamentais relacionados às linguagens de programação tanto de alto como de baixo nível. Observou-se, também, que o hardware consegue entender diretamente apenas a linguagem de máquina e que isto leva à necessidade de se traduzir os programas escrito em linguagens de alto nível para linguagem de máquina. Os programas que realizam esta função são chamados de **tradutores**.

Um programa escrito na linguagem original de alto nível utilizada é chamado de programa em **código fonte**. Por outro lado, um programa em linguagem de máquina gerado a partir de um processo de tradução é chamado de programa em **código objeto** (ou código de máquina).

Os programas tradutores podem ser de dois tipos: compiladores e interpretadores. Ambos têm a mesma função geral, mas diferem quanto ao modo de realizar o processo de tradução.

A diferença entre compiladores e interpretadores está no momento em que é realizada a tradução do programa:

- nos **compiladores**, a tradução e a execução dos programas são dois processos distintos. Primeiramente é realizada a tradução (chamada de compilação) e é gerado todo o código fonte. Então, o programa objeto pode ser executado quantas vezes for necessário.Os compiladores podem também realizar otimizações no código gerado,

- já os **interpretadores** traduzem cada estrutura de programação e imediatamente executam o código gerado. Pode-se notar que a execução de programas compilados é extremamente mais veloz que programas interpretados; isso porque os interpretadores traduzem e executam o programa fonte em tempo de execução. No entanto, são mais facilmente transportados para outros computadores com diferentes plataformas de hardware e software.

Programas com um compilador para uma plataforma específica normalmente não podem ser executados em computadores com plataforma diferente daquela para a qual foram gerados. Já programas interpretados utilizam diretamente o código fonte/intermediário, necessitando somente de um interpretador portado para a máquina-alvo.

7. CONSIDERAÇÕES FINAIS DA NOTA DE AULA:

Nesta aula de aula, vimos:

1. O que é um computador, a definição de uma máquina hipotética que irá nos fornecer um ambiente adequado para processar nossos algoritmos e o detalhamento de sua organização interna (ULA, UC, Memórias, etc.);
2. Foram tratados os conceitos fundamentais de uma linguagem algorítmica. Além da definição do que vem a ser um algoritmo, também foi possível conhecer suas características essenciais e técnicas básicas para sua construção (sequenciação de passos genéricos, refinamentos sucessivos, estruturas de decisão e iteração de blocos e/ou passos);
3. Viu-se que um algoritmo pode ser representado de diversas maneiras: estruturas narrativas, com fluxogramas e ainda com a definição de uma pseudolinguagem algorítmica; Também foram comentados os prós e contras com relação a cada uma das abordagens;
4. Para se ter uma idéia de como o computador opera sobre um conjunto de instruções, um algoritmo foi submetido à máquina hipotética para que o aluno pudesse observar com detalhes como cada uma das partes internas do computador interagem entre si durante a execução de um programa simples.
5. Conceitos acerca de linguagem de máquina (características, prós e contras, etc.), linguagem natural e linguagem de programação real de alto nível.
6. Finalmente, foram apresentados os conceitos e a importância de tradutores como compiladores e interpretadores.

8. CRÉDITOS E BIBLIOGRAFIA UTILIZADA PARA A GERAÇÃO DESTA NOTA DE AULA:

[1] EGYPTO, Cândido. **Lógica e Algoritmos**. CEFET-PB, 2003. (Livro-texto)
[2] PEREIRA, Frederico C. G. **Algoritmos**. CEFET-PB, 2004.