Departamento de Informática – UFPR

Algoritmos e Estruturas de Dados I

Prof.ª Dr.ª Mariane Cassenote

mariane@inf.ufpr.br

O modelo Von Neumann

Objetivos da aula

1. Mostrar como é o funcionamento dos computadores modernos em nível de máquina
2. Apresentar as limitações a que estamos sujeitos quando programamos

Aula elaborada com base na bibliografia da disciplina e
nos materiais dos Profs. Marcos Castilho e André Vignatti,
disponíveis no link

https://www.inf.ufpr.br/cursos/ci055/

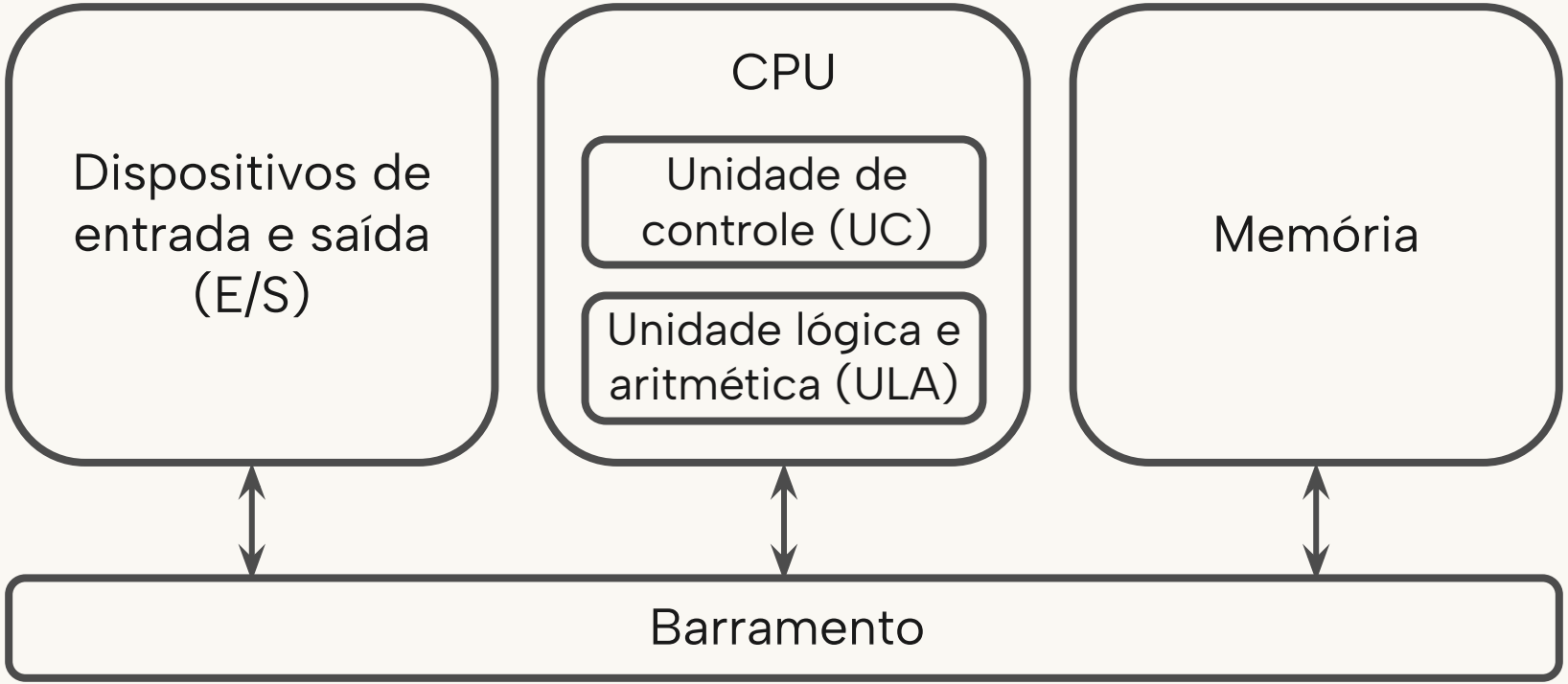
Histórico

- Os primeiros computadores são da década de 1930
  - Konrad Zuse construiu o primeiro computador (Z1) eletromecânico binário programável em 1936
- Computadores construídos para executar **programas específicos**
- Engenheiros **reconstruíam os computadores** para que eles fizessem cálculos diferentes
  - Esta tarefa consumia cerca de 3 semanas
  - Reprogramar não era uma tarefa trivial
- Entrada e saída era feita por **cartões perfurados**

Histórico

- Era preciso um novo modelo que facilitasse a programação
- A equipe de John Von Neumann implementou um modelo proposto por vários cientistas, incluindo Alan Turing, em que **tanto o PROGRAMA quanto os DADOS ficam em memória ao mesmo tempo**

Abstratamente



Como o computador executa o código?

1. Ele busca a instrução na memória
2. A Unidade de Controle (UC) decodifica (entende que é uma soma, por exemplo)
3. A Unidade Lógica e Aritmética (ULA) executa a operação
4. O resultado é guardado de volta na memória

É por isso que nossos algoritmos são sequenciais, executando um passo de cada vez

O Hardware

- Cada fabricante define qual o conjunto de instruções básicas para seu computador
- Este conjunto é implementado no circuito integrado e é tudo o que o computador sabe fazer!
- Processadores modernos implementam apenas algumas dezenas de instruções
- Programar significa realizar operações complexas usando apenas este pequeno conjunto de instruções

Exemplo: O computador da BCC

- BCC é a sigla da *Big Computer Company*
- Em termos teóricos tem o mesmo poder das suas concorrentes multinacionais
- O computador da BCC implementa apenas 9 (nove) instruções
- Ele tem uma memória RAM, um teclado e um monitor

Uma fotografia da memória RAM

Endereço	Conteúdo
0	1
1	54
2	2
3	1
4	50
5	4
6	7
7	46
8	4
9	47
10	46
11	46
12	7
13	48
14	4
15	49
16	50
17	48
18	3
19	51
20	47

Endereço	Conteúdo
21	49
22	6
23	52
24	51
25	3
26	53
27	46
28	52
29	5
30	55
31	53
32	54
33	8
34	55
35	2
36	56
37	46
38	52
39	5
40	57
41	56

Endereço	Conteúdo
42	54
43	8
44	57
45	9
46	33
47	2
48	76
49	67
50	76
51	124
52	14
53	47
54	235
55	35
56	23
57	78
58	243
59	27
60	88
61	12
62	12

Notação para endereços e conteúdos

- [0] =
- [[0]] =
- [0] + 1 =
- [0] + [1] =
- [0 + 1] =
- [[0] + [1]] =
- [[0] + 1] =

Denotamos por
[p] o **conteúdo** do
endereço p

O conjunto de instruções da BCC

Endereço	Conteúdo
0	1
1	54
2	2
3	1
4	50
5	4
6	7
7	46
8	4
9	47
10	46
11	46
12	7
13	48
14	4
15	49
16	50
17	48
18	3
19	51
20	47

Endereço	Conteúdo
21	49
22	6
23	52
24	51
25	3
26	53
27	46
28	52
29	5
30	55
31	53
32	54
33	8
34	55
35	2
36	56
37	46
38	52
39	5
40	57
41	56

Endereço	Conteúdo
42	54
43	8
44	57
45	9
46	33
47	2
48	76
49	67
50	76
51	124
52	14
53	47
54	235
55	35
56	23
57	78
58	243
59	27
60	88
61	12
62	12

- [0] =
- [[0]] =
- [0] + 1 =
- [0] + [1] =
- [0 + 1] =
- [[0] + [1]] =
- [[0] + 1] =

O conjunto de instruções da BCC

Cod.	Mnemônico	Instrução
1	load	Escreva em [p+1] o valor de [p+2]. Some 3 em p.
2	add	Escreva em [p + 1] a soma [[p + 2]] e [[p + 3]]. Some 4 em p.
3	sub	Escreva em [p + 1] a diferença [[p + 2]] e [[p + 3]]. Some 4 em p.
4	mult	Escreva em [p + 1] o produto [[p + 2]] por [[p + 3]]. Some 4 em p.
5	div	Escreva em [p + 1] a divisão [[p + 2]] por [[p + 3]]. Some 4 em p.
6	sqrt	Escreva em [p + 1] a raiz quadrada de [[p + 2]]. Some 3 em p.
7	read	Leia um número do teclado e guarde em [p + 1]. Some 2 em p.
8	write	Escreva [[p + 1]] na tela. Some 2 em p.
9	stop	Pare.

Ciclo de execução de instruções

- Comece com p valendo zero (p = 0)
- Interprete [p] de acordo com a tabela de instruções e só pare quando a instrução for uma ordem de parar (instrução 9)

Ciclo de execução de instruções

- Vamos nos colocar no lugar do computador e tentar entender como ele trabalha
- Vamos acompanhar o funcionamento dele a partir da “fotografia” da memória e do conjunto de instruções da BCC

Execução

Observações

- O computador não sabe o que está fazendo, somente obedece às instruções contidas na memória
- O ser humano percebe o que está acontecendo
- Existe uma grande diferença entre a **compreensão do ser humano** e a **mecânica do computador**
- O ser humano pode visualizar o processo de outra maneira!

Separando as instruções

Endereço	Instrução	Operando	Operando	Operando
0	1	54	2	
3	1	50	4	
6	7	46		
8	4	47	46	46
12	7	48		
14	4	49	50	48
18	3	51	47	49
22	6	52	51	
25	3	53	46	52
29	5	55	53	54
33	8	55		
35	2	56	46	52
39	5	57	56	54
43	8	57		
45	9			

Efeito desta nova visualização

- O computador não mudou o modo de operar
- Esta visualização apresentada apenas facilita o ser humano a perceber melhor o que o computador está fazendo
- Isto é um aspecto puramente humano, não computacional

Abstração de endereços de memória

- Os **endereços não são relevantes** para o ser humano
- Eles podem ser removidos e ainda podemos compreender o processo
- Para o computador, o endereço também não precisa ser fixo
- O importante é que o contador de programa inicie na primeira instrução

Abstração de endereços de memória

Endereço	Instrução	Operando	Operando	Operando
0	1	54	2	
3	1	50	4	
6	7	46		
8	4	47	46	46
12	7	48		
14	4	49	50	48
18	3	51	47	49
22	6	52	51	
25	3	53	46	52
29	5	55	53	54
33	8	55		
35	2	56	46	52
39	5	57	56	54
43	8	57		
45	9			

Abstração de endereços de memória

	Instrução	Operando	Operando	Operando
	1	54	2	
	1	50	4	
	7	46		
	4	47	46	46
	7	48		
	4	49	50	48
	3	51	47	49
	6	52	51	
	3	53	46	52
	5	55	53	54
	8	55		
	2	56	46	52
	5	57	56	54
	8	57		
	9			

Abstração do repertório de instruções

- Os seres humanos preferem usar nomes no lugar de números
- Por exemplo, eu sou mais conhecida pelo meu **nome**, embora para alguns sistemas eu seja o meu RG ou o meu CPF
- Logo, podemos **trocar os códigos das instruções por nomes**

Abstração do repertório de instruções

	Instrução	Operando	Operando	Operando
	load	54	2	
	load	50	4	
	read	46		
	mult	47	46	46
	read	48		
	mult	49	50	48
	sub	51	47	49
	sqrt	52	51	
	sub	53	46	52
	div	55	53	54
	write	55		
	add	56	46	52
	div	57	56	54
	write	57		
	stop	9		

Abstração do repertório de instruções

- Vamos agora abstrair as instruções com base nas tabelas seguintes

  X = [p+1]

  Y = [p+2]

  Z = [p+3]

1	load x y	X ← [y]
2	add x y z	X ← [y] + [z]
3	sub x y z	X ← [y] - [z]
4	mult x y z	X ← [y] * [z]
5	div x y z	X ← [y] / [z]
6	sqrt x y	X ← √[y]
7	read x	X ← v
8	write x	□ ← [x]
9	**stop**	●

Abstração do repertório de instruções

54 ← 2
50 ← 4
46 ← v
47 ← [46] * [46]
48 ← v
49 ← [50] * [48]
51 ← [47] - [49]
52 ← √[51]
53 ← [46] - [52]
55 ← [53] / [54]
□ ← [55]
56 ← [46] + [52]
57 ← [56] / [54]
□ ← [57]
●

Abstração dos endereços de memória

- Os **endereços de memória ainda estão sob a forma numérica**
- Mais uma vez, preferimos **nomes** para isto
- Um exemplo é o celular de hoje em dia, ligamos para um nome, não mais para um número (raramente o contrário)
- Então vamos dar **(bons) nomes** para as variáveis

Abstração dos endereços de memória

Endereço	Nome
54	dois
50	quatro
46	B
47	quadradoB
48	C
49	quadruploC
51	discriminante
52	raizdiscriminante
53	dobromenorraiz
55	menorraiz
56	dobromaiorraiz
57	maiorraiz

Abstração dos endereços de memória

```
dois            ← 2
quatro          ← 4
B               ← v
quadradoB       ← B * B
C               ← v
quadruploC      ← quatro * C
discriminante   ← quadradoB - quadruploC
raizdiscriminante ← √discrinante
dobromenorraiz  ← B - raizdiscriminante
menorraiz       ← dobromenorraiz / dois
□               ← menorraiz
dobromaiorraiz  ← B + raizdiscriminante
maiorraiz       ← dobromaiorraiz / dois
□               ← maiorraiz
●
```

Último grau de abstração direta

- Esta última forma de se visualizar o programa é a última forma de abstração direta
- A partir desta, é possível voltar à primeira forma (da fotografia da memória) apenas por traduções reversas
- A partir deste ponto, para vermos o programa na forma de alguma linguagem, precisamos dos **compiladores**

Linguagens Compiladas

- O compilador lê todo o código-fonte de uma vez e o traduz para linguagem de máquina (binário específico para um processador)
    - O resultado é um arquivo executável independente
- Processo: Código Fonte → Compilador → Arquivo Executável (Binário) → Execução.
- Exemplos: Pascal, C, C++, Rust, Go

Linguagens Compiladas

- Vantagens
  - Velocidade: O computador não precisa "traduzir" enquanto executa; o trabalho já foi feito
  - Otimização: O compilador pode otimizar o uso da memória e da ULA durante a tradução
- Desvantagens
  - Ciclo de desenvolvimento: Qualquer alteração exige uma nova compilação
  - Dependência de plataforma: Um executável gerado para Windows (Intel) não rodará no Linux (ARM) sem ser recompilado

Linguagens Interpretadas

- Não existe um arquivo executável binário. Um programa chamado intérprete lê o código-fonte linha por linha e executa as instruções "na hora"
  - Processo: Código Fonte → Intérprete (tradução e execução simultâneas)
  - Exemplos: Python, JavaScript, PHP, Ruby

Linguagens Interpretadas

- Vantagens
  - Portabilidade: O mesmo código roda em qualquer sistema que tenha o intérprete instalado
  - Facilidade de Debug: É mais fácil testar pequenos trechos de código rapidamente
- Desvantagens
  - Desempenho: É geralmente mais lenta, pois o computador gasta tempo traduzindo cada linha enquanto o programa roda

Linguagens Híbridas

- Algumas linguagens modernas, como Java e C#, usam uma abordagem mista:
  - O código é compilado para uma linguagem intermediária (chamada Bytecode)
  - Esse Bytecode é então interpretado ou compilado "na hora" por uma Máquina Virtual (como a JVM)

Linguagens

- As linguagens de programação são baseadas em **gramáticas rígidas** e portanto o formato dos texto é bastante restrito
- As linguagens seguem diferentes **paradigmas**, que basicamente definem seu grau de restrição
    - Linguagens estruturadas, funcionais, lógicas, orientadas a objetos, etc.

Linguagens

- As linguagens chamadas de **‘alto nível**’ são aquelas que permitem ao ser humano um alto grau de compreensão do texto que está escrito, embora ainda longe da nossa chamada língua natural
- Um dos motivos da rigidez das linguagens de programação é a **eliminação de ambiguidade** dos textos (linguagens livres de contexto)
- A seguir veremos nosso programa escrito em uma linguagem de alto nível (procedural e imperativa) chamada Pascal

Versão do programa em Pascal

```
program Bhaskara (input,output);
var b,c,raizdiscriminante : real;

begin
  read (b);
  read (c);
  raizdiscriminante := sqrt (b*b - 4*c);
  write ((b - raizdiscriminante)/2);
  write ((b + raizdiscriminante)/2);
end.
```

O que é programar?

- Programar um computador em alto nível é, basicamente, conhecer e dominar uma notação através da qual textos (ou **programas fonte**) são traduzidos para programas **executáveis**
- Programar significa concatenar as instruções disponíveis dentro de um repertório a fim de **transformar dados de entrada em dados de saída** para resolver um problema

O que se precisa ter para programar?

- Ter a disposição um **editor de textos** (ASCII) para codificar o algoritmo em forma de programa fonte
- Ter à disposição um **compilador** para a linguagem escolhida para transformar automaticamente um programa fonte em um programa executável
- Evidentemente, ter habilidade de escrever textos, que são **programas que resolvem problemas**

Evolução e alternativas

- **Inteligência Artificial** voltada para linguagens de ainda mais alto nível, por exemplo linguagens sensíveis ao contexto
- **Computação Quântica**: escapa do modelo von Neumann, podem dar origem a outros tipos de linguagens

Exercícios

- Fazer os exercícios do capítulo 4 do livro

  https://www.inf.ufpr.br/marcos/livro_alg1/livro_alg1.pdf

Fim do tópico

- O conteúdo desta aula está no livro no capítulo 4
- Na próxima aula veremos os conceitos básicos das linguagens de programação

Departamento de Informática – UFPR

Algoritmos e Estruturas de Dados I

Prof.ª Dr.ª Mariane Cassenote

mariane@inf.ufpr.br

O modelo Von Neumann