

MAC 113 — Introdução à Ciência da Computação

Aula 8

Nelson Lago

1º/2025



Previously on MAC 113...

Tipos de repetição

- **Dois tipos fundamentais de repetição**

- ① Repetições até atingir um resultado

- » *Encontrar o próximo primo*
 - » *Reiniciar o jogo até o usuário escolher “sair”*
 - » ...

- ② Repetições sobre os elementos de um conjunto

- » *Apresentar todos os pixels de uma foto na tela*
 - » *Trocar todas as letras de um texto para maiúsculas*
 - » ...

Repetições e condições

- Em ambos os casos, “algo” precisa acontecer para indicar que as repetições chegaram ao fim (o objetivo foi atingido ou todos os elementos do conjunto já foram processados)

(ok, às vezes queremos repetir indefinidamente, mas vamos ignorar isso por enquanto)

- **As repetições são controladas por algum tipo de condição baseada no estado de uma *variável***

```
chute <- readline("Adivinhe qual é minha cor favorita: ")
while (chute != "rosa choque") {
  chute <- readline("Errou! Tente novamente: ")
}
cat("Acertou!", "\n")
```

Repetições e condições

- A **condição** é um **booleano**

```
chute <- readline("Adivinhe qual é minha cor favorita: ")
while (chute != "rosa choque") {
  chute <- readline("Errou! Tente novamente: ")
}
cat("Acertou!", "\n")
```

```
chute <- readline("Adivinhe qual é minha cor favorita: ")
if (chute != "rosa choque") {
  chute <- readline("Errou! Tente novamente: ")
  vaidinovo!!!!
}
cat("Acertou!", "\n")
```

Partes mínimas de um laço

- **Um laço correto precisa**

- ▶ Inicializar a variável de controle antes do início do laço
- ▶ Verificar a condição adequada a cada iteração para que as repetições aconteçam o número correto de vezes
- ▶ Alterar o valor da variável de acordo com a lógica do programa (no mínimo, na última iteração) para garantir que o laço termine

```
chute <- readline("Adivinhe qual é minha cor favorita: ")
while (chute != "rosa choque") {
  chute <- readline("Errou! Tente novamente: ")
}
cat("Acertou!", "\n")
```

Repetições até atingir um resultado

- **Caso comum:**

- ▶ branco sujo
- ▶ amarelo icterícia
- ▶ roxo hematoma
- ▶ **rosa choque**

*A **variável** muda em todas as iterações, mas a **condição** só muda na última iteração*

- **Caso sortudo:**

- ▶ **rosa choque**

***Nenhuma** iteração, então nem a variável nem a condição mudam*

- **Caso absurdo:**

- ▶ roxo hematoma
- ▶ roxo hematoma
- ▶ roxo hematoma
- ▶ **rosa choque**

*A variável **e** a condição só mudam na última iteração*

A variável sempre precisa mudar (ao menos) na última iteração!

Exercício

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos (“de trás para a frente”). Por exemplo, $6437 \rightarrow 7346$

```
n <- as.integer(readline("Digite um número natural: "))
invertido <- 0
tmp <- n
while (tmp > 0) {
  invertido <- invertido * 10
  invertido <- invertido + tmp %% 10
  tmp <- tmp %% 10
}
cat(n, "invertido é", invertido, "\n")
```

Exercício — adivinhando números 1/4

Faça um programa que escolhe um número de 1 a 10 e pede para o usuário tentar adivinhá-lo até que o usuário tenha sucesso (repetições até atingir um resultado)

```
main <- function() {  
  n <- sample(1:10, 1)  
  chute <- as.integer(readline("Escolhi um número de 1 a 10; tente adivinhá-lo! "))  
  while (chute != n) {  
    chute <- as.integer(readline("Errou! Tente de novo: "))  
  }  
  cat("Parabéns, você acertou!", "\n")  
}  
main()
```

And now for something completely different

Prelúdio

O acumulador

- Para resolver uma conta no papel, fazemos “afirmações”, como $1 + 2 - 3 + 4$
- Reescrevemos a afirmação conforme vamos realizando as diversas operações
 - ▶ Ou seja, não é necessário usar borracha
 - ▶ O “histórico” dos passos é visível
 - ▶ Podemos escolher a ordem em que vamos realizar os passos
 - » *Porque a expressão completa está sempre visível*

$$\begin{array}{r} 2 \times (3 + 4) - 5 + 6 - 7 \\ \hline 8 \end{array}$$

$$\begin{array}{r} 2 \times (3 + 4) + 1 - 7 \\ \hline 8 \end{array}$$

$$\begin{array}{r} 2 \times 7 + 1 - 7 \\ \hline 8 \end{array}$$

$$\begin{array}{r} 2 \times 7 - 6 \\ \hline 8 \end{array}$$

$$\begin{array}{r} 14 - 6 \\ \hline 8 \end{array}$$

$$\begin{array}{r} 7 - 3 \\ \hline 4 \end{array}$$

$$\begin{array}{r} 4 \\ \hline 4 \end{array}$$

$$1$$

O acumulador

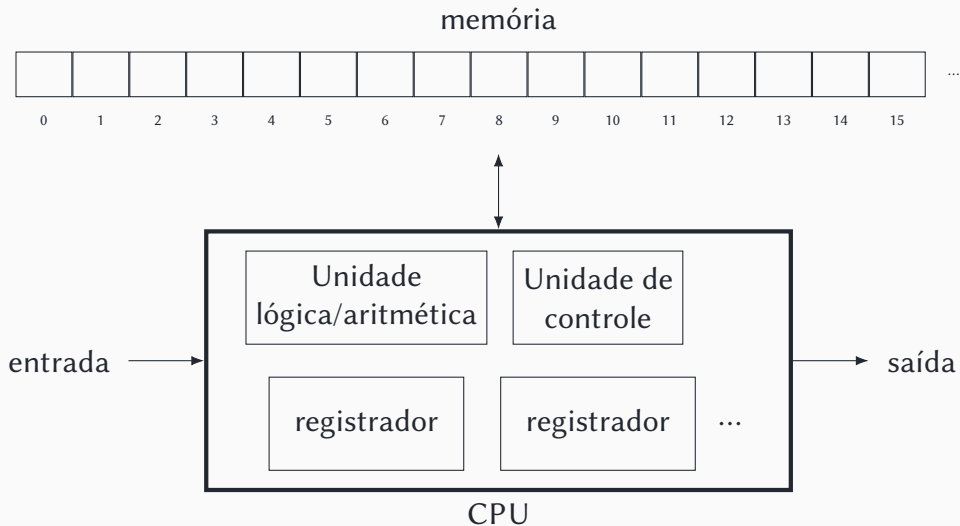
- **Mas não é assim que fazemos contas “de cabeça”**
 - ▶ Executamos os passos sequencialmente
 - ▶ A cada passo, guardamos o resultado na memória e processamos esse resultado com o próximo operando
 - » *Ou seja, temos sempre um resultado intermediário*
 - » *Esse resultado intermediário é um dos operandos da próxima operação*
 - » *Ele é “jogado fora” e substituído por um novo resultado intermediário a cada passo (no papel, usaríamos a borracha)*
 - » *Após o último passo, ele é o resultado final*
- **O mesmo acontece com ábacos, calculadoras de bolso... e o computador!**
- Chamamos esse “espaço de rascunho” de **acumulador**

Soltando os parafusos

Recursos essenciais do computador que vimos até agora:

- **Sequência de instruções (programa) definida pelo usuário**
- **Leitura e escrita de dados (teclado e tela)**
- **Operações matemáticas (soma, divisão etc.) + acumulador**
- **Números inteiros**
- **Referências a dados na memória**
- **Condicionais e laços (caso especial de condicionais)**

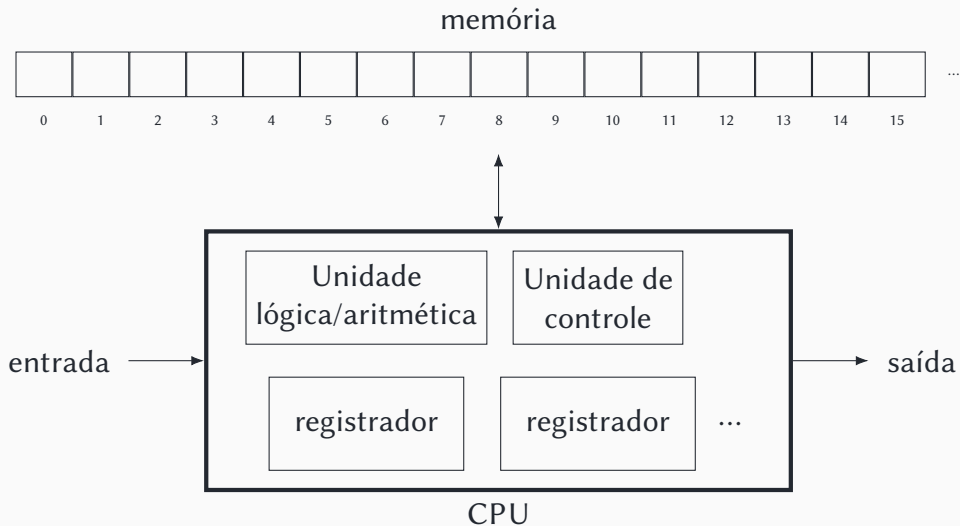
Arquitetura de von Neumann



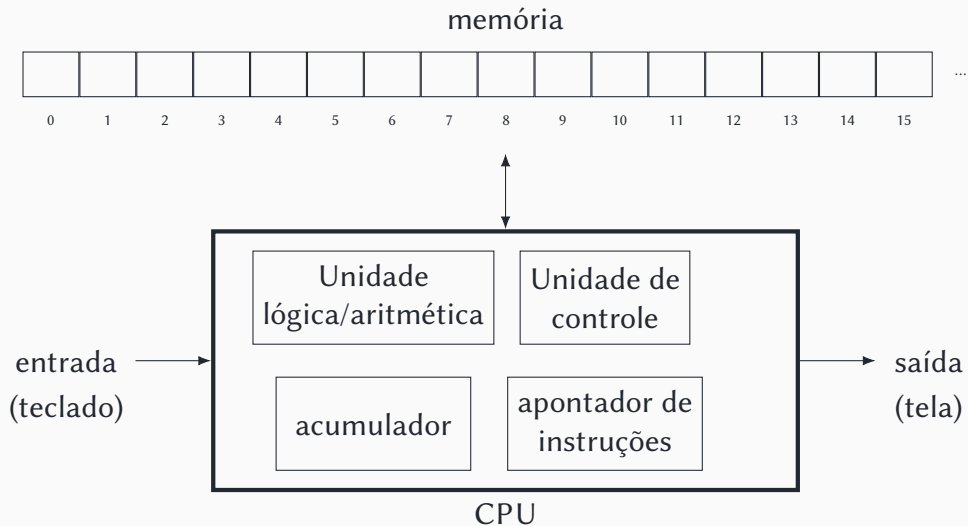
O computador HIPO

O computador HIPO (“hipotético”) é um computador fictício criado pelo prof. Valdemar Setzer para o ensino de conceitos básicos de computação, descrito em www.ime.usp.br/~vwsetzer/HIPO/hipo-descr.html.

Arquitetura de von Neumann



O computador HIPO



O computador HIPO

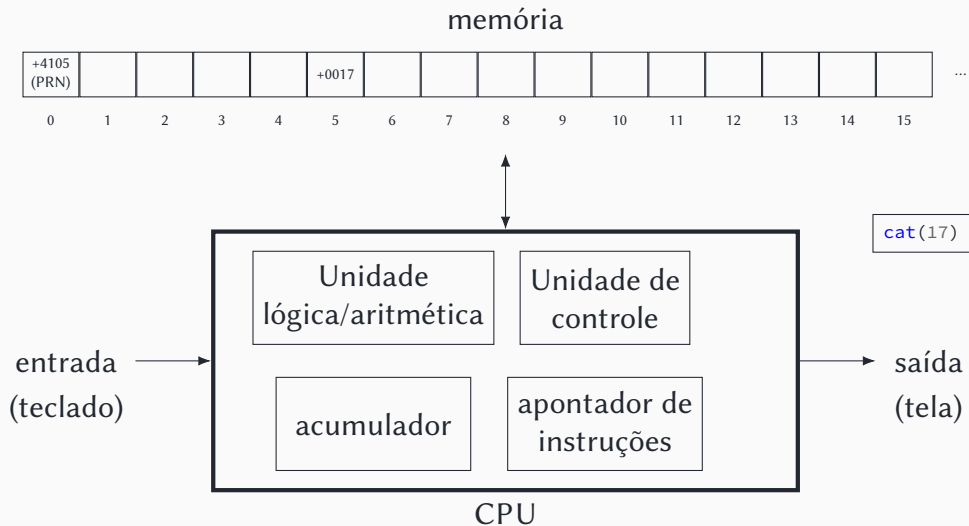
- O computador HIPO tem 100 posições de memória (0–99), que armazenam números no formato sDDDD, onde s é o sinal (+ ou –) e D são dígitos de 0 a 9. Portanto, ele trabalha com números no intervalo -9999 a +9999.
- Uma posição de memória pode ser interpretada como uma instrução. Nesse caso, o formato é +IIAA, onde II é a instrução e AA é o endereço de memória ao qual aplicar a instrução (00–99).
 - ▶ (instruções que dependem de dois operandos, como a soma, processam o endereço de memória fornecido em conjunto com o acumulador)

O computador HIPO

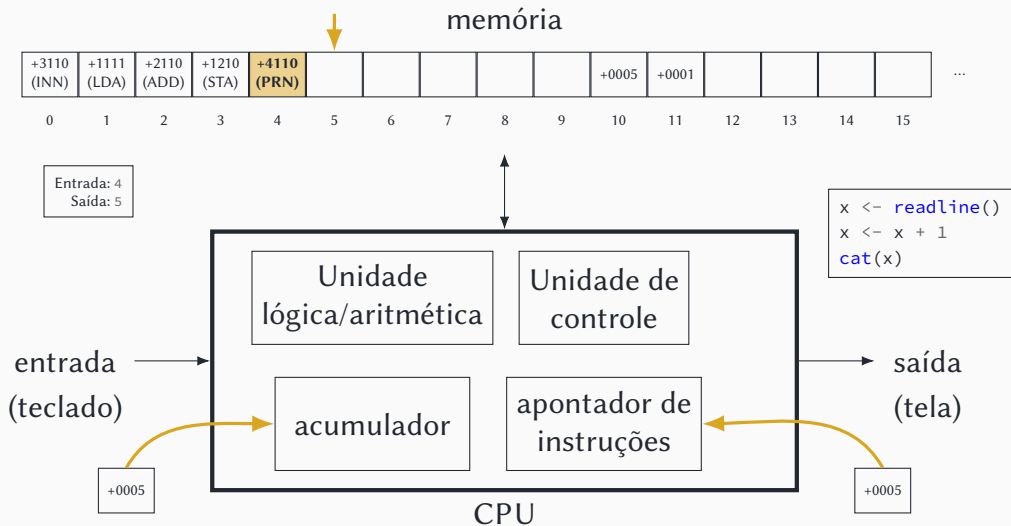
11	LDA	Load from memory to accumulator
12	STA	Store in memory from accumulator
21	ADD	Add memory and accumulator (result is left in accumulator)
22	SUB	Same, but subtraction
23	MUL	Same, but multiplication
24	DIV	Same, but division
25	REM	Same, but remainder
29	REV	Revert sign (+/-) of the accumulator
31	INN	Load from keyboard to memory
41	PRN	Show from memory to screen

Instruções reconhecidas pelo computador HIPO

Primeiro programa para o computador HIPO



Segundo programa para o computador HIPO



O computador HIPO

11	LDA	Load from memory to accumulator
12	STA	Store in memory from accumulator
21	ADD	Add memory and accumulator (result is left in accumulator)
22	SUB	Same, but subtraction
23	MUL	Same, but multiplication
24	DIV	Same, but division
25	REM	Same, but remainder
29	REV	Revert sign (+/-) of the accumulator
31	INN	Load from keyboard to memory
41	PRN	Show from memory to screen

Instruções reconhecidas pelo computador HIPO

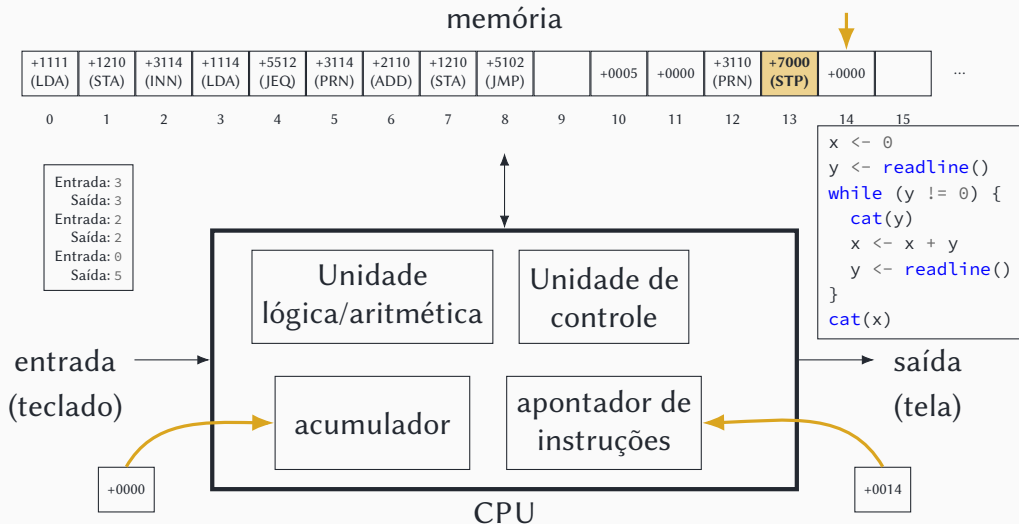
O computador HIPO

11	LDA	Load from memory to accumulator
12	STA	Store in memory from accumulator
21	ADD	Add memory and accumulator (result is left in accumulator)
22	SUB	Same, but subtraction
23	MUL	Same, but multiplication
24	DIV	Same, but division
25	REM	Same, but remainder
29	REV	Revert sign (+/-) of the accumulator
31	INN	Load from keyboard to memory
41	PRN	Show from memory to screen

51	JMP	Jump to given address
55	JEQ	Same, but only if the accumulator is zero
70	STP	Stop

Instruções reconhecidas pelo computador HIPO

Terceiro programa para o computador HIPO



O computador HIPO

11	LDA	Load from memory to accumulator
12	STA	Store in memory from accumulator
21	ADD	Add memory and accumulator (result is left in accumulator)
22	SUB	Same, but subtraction
23	MUL	Same, but multiplication
24	DIV	Same, but division
25	REM	Same, but remainder
29	REV	Revert sign (+/-) of the accumulator
31	INN	Load from keyboard to memory
41	PRN	Show from memory to screen

51	JMP	Jump to given address
55	JEQ	Same, but only if the accumulator is zero
70	STP	Stop

Instruções reconhecidas pelo computador HIPO

O computador HIPO

11	LDA	Load from memory to accumulator
12	STA	Store in memory from accumulator
21	ADD	Add memory and accumulator (result is left in accumulator)
22	SUB	Same, but subtraction
23	MUL	Same, but multiplication
24	DIV	Same, but division
25	REM	Same, but remainder
29	REV	Revert sign (+/-) of the accumulator
31	INN	Load from keyboard to memory
41	PRN	Show from memory to screen

50	NOP	Do nothing
51	JMP	Jump to given address
55	JEQ	Same, but only if the accumulator is zero
53	JDZ	Same, if not zero
57	JGE	Same, if zero or more
54	JGT	Same, if greater than zero
52	JLE	Same, if zero or less
56	JLT	Same, if less than zero
70	STP	Stop

Instruções reconhecidas pelo computador HIPO

Linguagens de programação

- Linguagem de máquina

+1111 +1210 +3114 +1114 +5512 +3114 +2110 +1210 +5102...

- Linguagem assembly

```
LDA 11
STA 10
INN 14
JEQ 12
PRN 14
ADD 10
STA 10
JMP 02
...
```

- Linguagens de alto nível

```
x <- 0
y <- readline()
while (y != 0) {
    cat(y)
    x <- x + y
    y <- readline()
}
cat(x)
```

Linguagens de programação

- Existe um programa responsável por “traduzir” a linguagem assembly para a linguagem de máquina:
 - ▶ O *assembler*
- Existe um programa responsável por “traduzir” as linguagens de alto nível para a linguagem de máquina:
 - ▶ O *compilador* da linguagem
 - ▶ O *interpretador* da linguagem
 - » (em algumas linguagens, o interpretador utiliza técnicas “híbridas”, como compilação para bytecodes, compilação JIT ou compilação AoT)

- **Em 1936 (muito antes de o primeiro computador no sentido moderno ser construído), Alan Turing também inventou um computador hipotético, muito mais simples que o computador HIPO**
 - ▶ “máquina de Turing”
- **O objetivo era ter um modelo matemático o mais simples possível para a computabilidade**
 - ▶ (ela foi usada para demonstrar alguns teoremas fundamentais da área)

- Um “computador” (ou linguagem de programação) capaz de simular uma máquina de Turing é “Turing-completo”
- Um sistema que *não* é Turing-completo é “menos poderoso” que uma máquina de Turing
 - ▶ (ou seja, é incapaz de resolver parte dos problemas que uma máquina de Turing pode resolver)

- Um dos requisitos mais “difíceis” para um sistema ser Turing-completo é a capacidade de executar condicionais e laços
 - ▶ Várias calculadoras e “computadores” foram criados a partir do século XVIII, mas nenhum era Turing-completo por conta disso
 - » *(laços e condicionais não eram necessários para os objetivos desses dispositivos)*
- O primeiro computador Turing-completo funcional foi o ENIAC, de 1946
- O primeiro computador Turing-completo inventado mas não construído foi a máquina analítica de Charles Babbage, de 1837

- **Turing postulou que qualquer “computador”, por mais sofisticado que seja, não é mais expressivo que sua criação (conjectura/tese de Church-Turing)**
 - ▶ E que, portanto, qualquer computador pode ser simulado por uma máquina de Turing
- **Aceitando-se essa hipótese, todo computador que é Turing-completo (pode simular uma MT) é equivalente a uma máquina de Turing (pode ser simulado por uma MT)**

International Obfuscated C Code Contest

[illegible]

3.141

(www.ioccc.org/years-spoiler.html#1988_westley)

Linguagem Brainfuck

```
+++++ +++           Set Cell #0 to 8
[
  >++++             Add 4 to Cell #1; this will always set Cell #1 to 4
                    as the cell will be cleared by the loop
  [
    >+              Add 4*2 to Cell #2
    >+              Add 4*3 to Cell #3
    >+              Add 4*3 to Cell #4
    >+              Add 4 to Cell #5
    <<<<-          Decrement the loop counter in Cell #1
  ]               Loop till Cell #1 is zero
  >+              Add 1 to Cell #2
  >+              Add 1 to Cell #3
  >-              Subtract 1 from Cell #4
  >>+             Add 1 to Cell #6
  [<]             Move back to the first zero cell you find; this will
                    be Cell #1 which was cleared by the previous loop
  <-              Decrement the loop Counter in Cell #0
]               Loop till Cell #0 is zero
```

The result of this is:

```
Cell No :  0  1  2  3  4  5  6
Contents:  0  0 72 104 88 32  8
Pointer :  ^
```

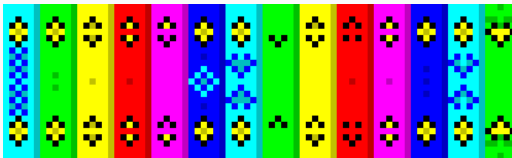
```
>>.               Cell #2 has value 72 which is 'H'
>---.             Subtract 3 from Cell #3 to get 101 which is 'e'
+++++ ++.++++.    Likewise for 'llo' from Cell #3
>>.               Cell #5 is 32 for the space
<-.               Subtract 1 from Cell #4 for 87 to give a 'W'
<.               Cell #3 was set to 'o' from the end of 'Hello'
+++----- -.----- ---. Cell #3 for 'rl' and 'd'
>>+.              Add 1 to Cell #5 gives us an exclamation point
>+.              And finally a newline from Cell #6
```

```
++++++[>++++[>+>+>+>+>+<<<<-]>+>->->+ [<]<-]>+.
>---.+++++.+.+.+.>+.<-.<+.+-----.----->+.>+.
```

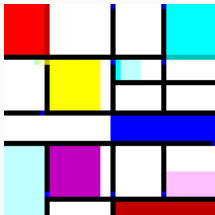
Hello World!

(esolangs.org/wiki/Brainfuck)

Linguagem Piet



Dois programas que imprimem “Hello, World!”



Programa que imprime “Piet” (em homenagem a Piet Mondrian)