

Sistemas Digitais

NEANDER

Etapa 04

Gabriela Stein

Ano letivo 2024

# Introdução

# NEANDER

## Computador Teórico

- Voltado para o ensino de conceitos introdutórios de Organização e Arquitetura de Computadores
  - UFRGS
  - Raul Fernando Weber †2018
- Utilizado também como projeto final de Sistemas Digitais
  - Unioeste
- Inspirado no Computador IAS (Von Neumann)

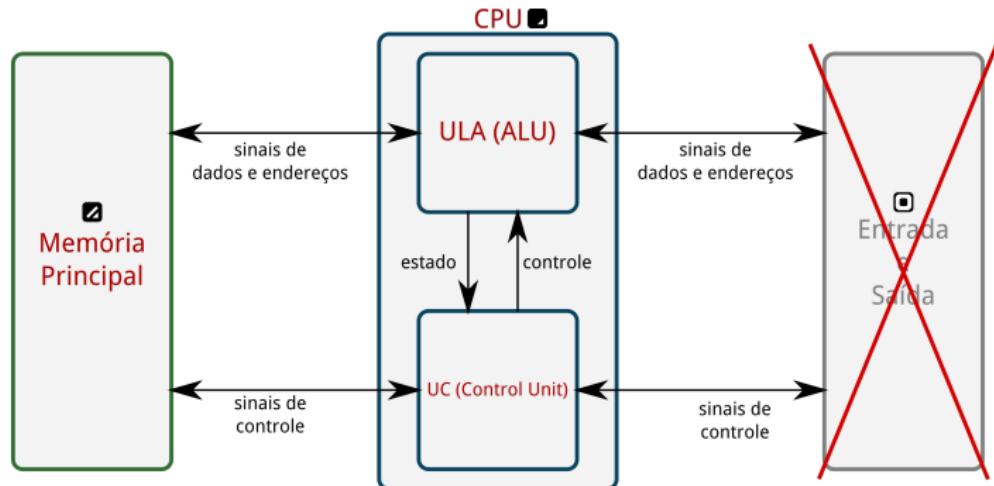
*“É tão simples que você, sem muito esforço, pode programá-lo e projetá-lo”*

- Todo este material é baseado no livro  
WEBBER, Raul Fernando, Fundamentos de Arquitetura de Computadores. Porto Alegre, Bokman, 4 Edição. 2012.

# NEANDER

Computador Teórico?

- Arquitetura de Vonn Neumann sem Entrada e Saída



- E como funciona desse jeito?
  - à moda antiga!

# NEANDER - Funcionamento geral

- Memória Principal é alimentada com Instruções e Dados
  - Antigamente, por cartões perfurados
  - NEANDER? Por carregamento de arquivo no simulador
- Unidade de Controle
  - Busca Instrução na posição  $i$
  - Decodifica Instrução
  - Aciona Sinais de Controle
    - Aciona buscas em memória
    - Aciona ULA
    - Até que a Instrução seja executada por completo
  - Incrementa  $i$
  - repete até o programa encerrar
    - Instrução especial de controle

# NEANDER - Características I

Tipos de dados:

- Computadores de Propósito Geral (CPG) contém vários tipos de dados
  - Inteiros (4 bytes), Real (4 ou 8 bytes), Caracteres (1 byte), etc
- NEANDER: somente inteiro sinalizado de 1 byte
  - Representação em Complemento de Dois

Lagura dos dados:

- CPG: variável de 1 byte até 16 bytes (AVX 512)
- NEANDER: sempre 1 byte

# NEANDER - Características II

## Identificadores de Variáveis

- CPG: yep, good! basta usar algum texto com restrições
  - identificador “esconde” endereço do programador

```
int nomeVariavel;           // identificador válido
float outraVariavel;        // idem
char 5&*noooooooooo;       // identificador inválido
```

- NEANDER: não é tão simples!
  - Não contém identificadores
  - Cada variável é referenciada pelo seu endereço

```
// exemplo didático com sintaxe não aplicável
ac = mem[128]
// ac recebe o conteúdo da memória na posição 128
```

# NEANDER - Características III

## Modos de Endereçamento

- CPG: imediato, direto, indireto, etc

```
a = 10;           // imediato
b = idade;        // direto
c = *peso;         // indireto
d = istoehumvetor[3]; // indexado
```

- NEANDER: somente direto

```
// exemplo didático com sintaxe não aplicável
ac = mem[128]      // direto
// 'ac' recebe conteúdo da memória na posição 128
// 'ac' não é identificador! aguarde!
```

# NEANDER - Características IV

## Quantidade de instruções

- CPG: depende de como o observador quantifica!
  - de acordo com Fabian Giesen, IA-32 contém  $\approx 1503$  instruções  
Fonte: [Mark J Charney - Intel X86 Encoder Decoder Library](#)  
de AAA to XTEST  
Quer saber mais sobre instruções da arquitetura IA32-64?  
[Intel\(R\) 64 and IA-32 Architectures Software Developer's Manual](#)  
(cuidado, pdf de +5000 páginas)

## ■ NEANDER:

- somente 11 instruções:
  - 4 de controle
  - 3 lógicas
  - 2 de transferência
  - 1 aritmética
  - 1 *meio que* de controle que não faz nada!

# NEANDER - Características V

## Formato das instruções

- CPG: depende da arquitetura
  - IA-32-64: uma pequena salada que pode variar entre 1 até 15 bytes
    - Aguardem LM e OAC!
- NEANDER:
  - 1 byte
    - Instruções com operando implícito NOP, NOT e HLT
  - 2 bytes
    - Instruções com operando end
      - 1º byte: código da operação
      - 2º byte: operando end

# NEANDER - Características VI

## Memória

- CPG: sistema hierárquico composto de:
  - Nível 0: Registradores (pequeno e rápido)
    - 8 (IA-32) ou 16 (IA32-64) registradores de Propósito Geral
    - 8 (IA-32) ou 16 (IA32-64) registradores de Propósito Específico
    - Registradores de controle e acesso à barramentos não visíveis
  - Nível 1: Caches
  - Nível 2: Memória Principal
    - Tamanhos variam de 2GiB (office) até 64 GiB (gamer)
  - Nível 3: Memória Secundária (enorme e lenta)
- NEANDER
  - Apenas 1 único Registrador chamado AC
  - Memória Principal de 256 posições com 1 byte cada uma
    - Acesso byte-a-byte

# Programação NEANDER

# NEANDER

## Características:

- Não tem Entrada e Saída
  - Instruções e Dados são carregados na Memória Principal
- Dados inteiros em Complemento de Dois de 1 byte
- Sem identificador de variáveis e Endereçamento Direto
  - Usa-se o endereço do local da memória que contém o dado
  - Acesso byte-a-byte
- 1 único registrador chamado AC
- 11 instruções
  - formato com 1 ou 2 bytes

# Hello World I

“Hello World”:

- ??

```
a = a + b;
```

- Linguagem NEANDER

- Considerações:

- a* está na posição *mem[128]*

- b* está na posição *mem[129]*

- LDA end* instrução que carrega *AC* com *mem[end]*

- STA end* instrução que armazena *AC* em *mem[end]*

- ADD end* instrução que soma *AC* com *mem[end]* e salva em *AC*

- HLT* instrução que encerra um programa (return 0;)

# Hello World II

```
a = a + b;
```

## ■ Linguagem NEANDER

- Considerações:

- a* está na posição *mem[128]*

- b* está na posição *mem[129]*

- LDA end: instrução que carrega *AC* com *mem[end]*

- STA end: instrução que armazena *AC* em *mem[end]*

- ADD end: instrução que soma *AC* com *mem[end]* e salva em *AC*

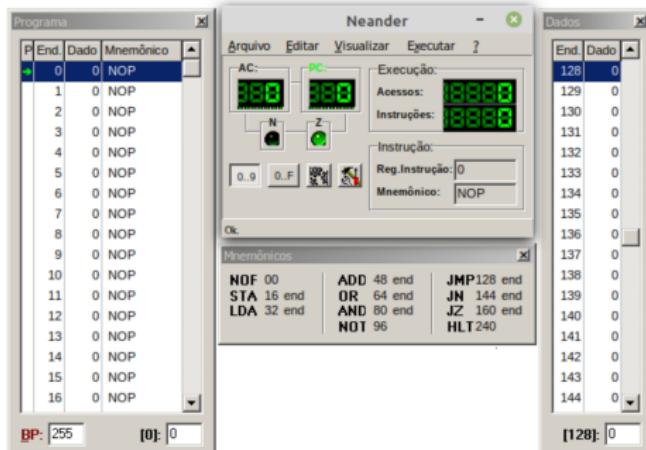
- HLT : instrução que encerra um programa (return 0;)

```
1      LDA 128 ; carrega 'a' em AC
2      ADD 129 ; adiciona 'b' em AC
3      STA 128 ; salva AC na mem[128], que é 'a'
4      HLT      ; encerra o programa
```

# Programação no WNEANDER

## WNEANDER

- Simulador do NEANDER descrito no livro do Weber



- Windows
- Linux utilizando Wine

A utilização do simulador será apresentado em aula, isto é, este documento não contém um passo-a-passo!

# Programação NEANDER

## Conjunto de Instruções:

Tipo	Mnemônico	Descrição	Comentário
c	NOP	<i>no-operation</i>	Não executa ações sobre Dados
t	STA end	MEM[end] $\leftarrow$ AC	STORE: armazena AC na memória
t	LDA end	AC $\leftarrow$ MEM[end]	LOAD: carrega AC com conteúdo da memória
a	ADD end	AC $\leftarrow$ AC + MEM[end]	Adição entre AC e conteúdo da memória
l	OR end	AC $\leftarrow$ AC or MEM[end]	Ou-lógico entre AC e conteúdo da memória
l	AND end	AC $\leftarrow$ AC and MEM[end]	E-lógico entre AC e conteúdo da memória
l	NOT	AC $\leftarrow$ not(AC)	Não-lógico de AC
c	JMP end	PC $\leftarrow$ end	Altera valor de PC para <b>end</b>
c	JN end	PC $\leftarrow$ end SE NF==1	Altera valor de PC para <b>end</b> somente se NF for 1
c	JZ end	PC $\leftarrow$ end SE ZF==1	Altera valor de PC para <b>end</b> somente se ZF for 1
c	HLT	HALT	Encerra execução do programa

c: controle

t: transferência

l: lógica

a: aritmética

# Programação NEANDER

## FLAGS:

- $NF$  e  $ZF$
- Operações que resultam em alteração de AC
  - Se  $AC$  for um número negativo:  $NF = 1$
  - Se  $AC$  for zero:  $ZF = 1$
- FLAGS são utilizadas para decidir Saltos-Condicionais
  - Instrução JN end
    - Salta para *end* se  $NF == 1$   
Caso contrário, executa para próxima instrução
  - Instrução JZ end
    - Salta para *end* se  $ZF == 1$   
Caso contrário, executa para próxima instrução

# Introdução - Exercícios

# Exercícios com simulador WNEANDER

Para compreensão do funcionamento do NEANDER

- Como implementar em VHDL sem saber o que o NEANDER faz?

NEANDER - Lista de Exercício 01

- Arquivo NEANDER-Lista01-Exercícios wnenader.pdf
- Na Pasta Lista de Exercícios na Equipe MSTeams.

# Arquitetura e Organização NEANDER

# Arquitetura NEANDER

Arquitetura:

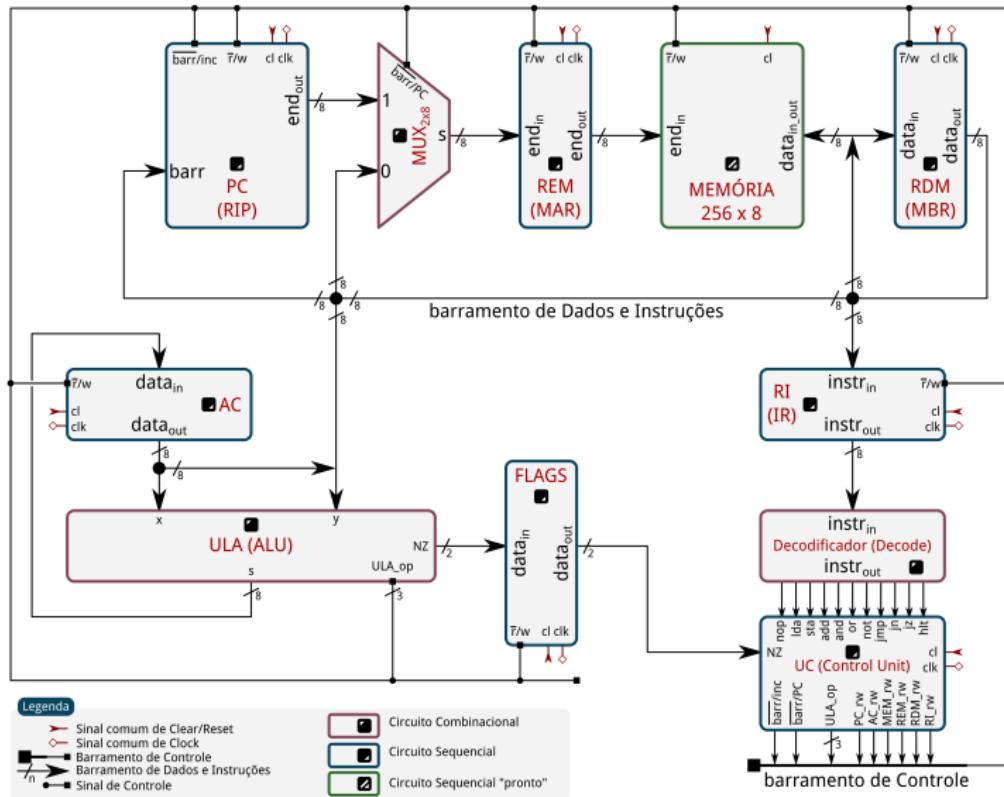
- Atributos visíveis ao programador
  - Conjunto de Instruções
  - Largura e Tipos de Dados
  - Modos de Endereçamento
  - Entre outros
- É o que a máquina fornece ao programador para que este escreva os programas.
- Arquitetura NEANDER foi discutida na Seção Introdução

# Organização NEANDER

Organização:

- Atributos transparentes ao programador
  - Módulos internos
  - Interconexões
- É como a máquina executa e interconecta os módulos internos de modo a implementar a arquitetura

# Organização NEANDER (00-baseNeander.png)

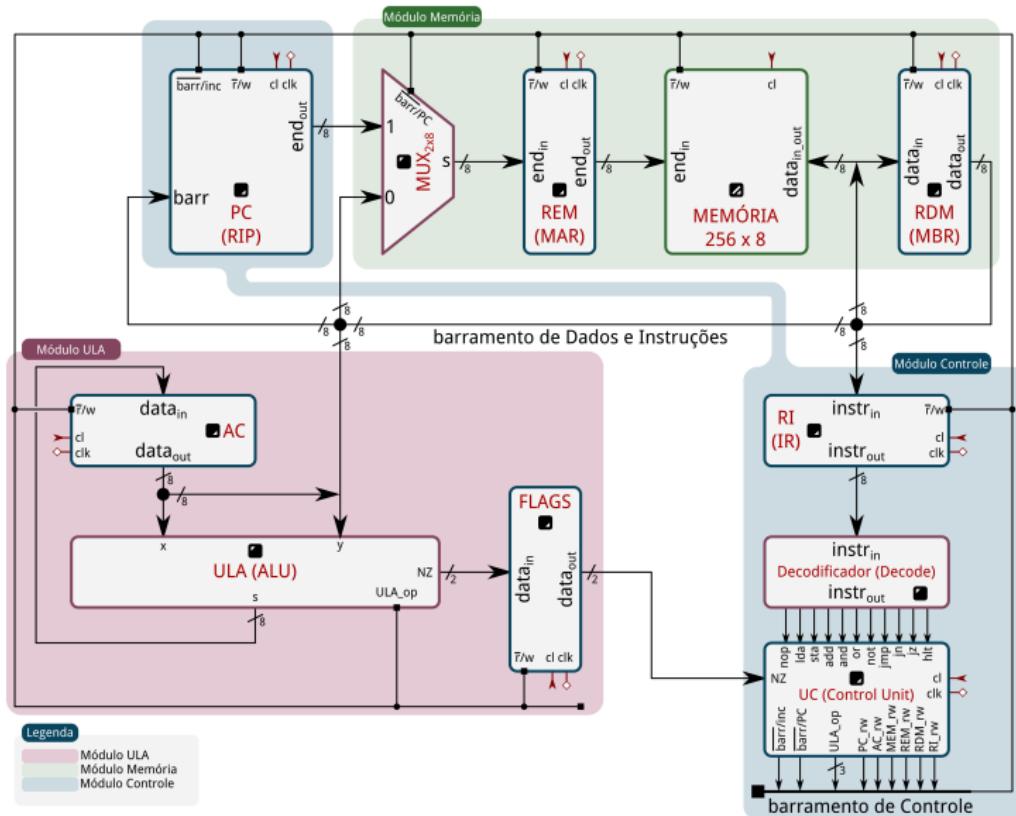


# Organização NEANDER

## Organização:

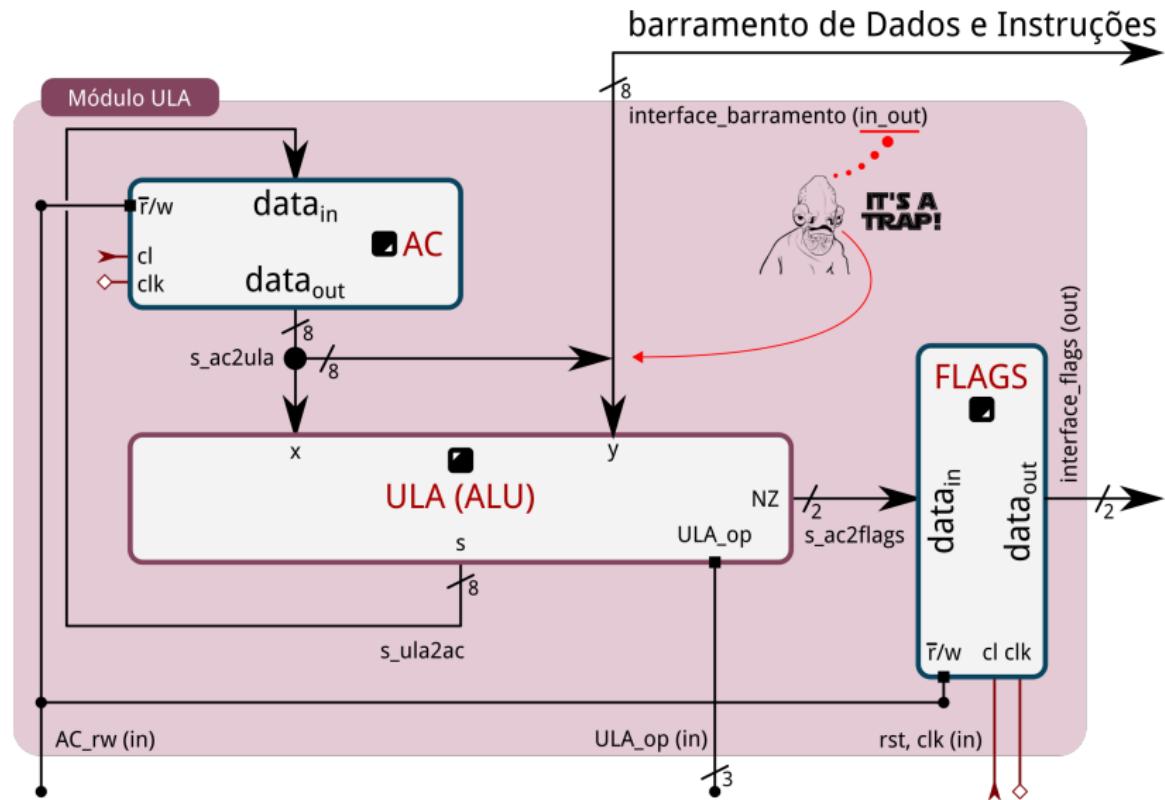
- Organização NEANDER será apresentada em 3 Seções:
  - Módulo ULA
  - Módulo Memória
    - Interconexão com Módulo ULA
  - Módulo de Controle
    - Interconexão com Módulo ULA e Módulo Memória

# Organização NEANDER (01-modulosNeander.png)



# Organização NEANDER - ULA

# Módulo ULA



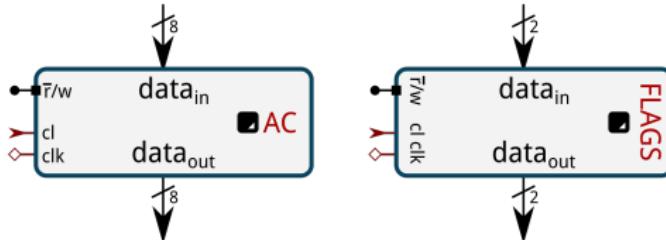
# Módulo ULA - Componentes I

## ■ Registrador AC

- sinal  $\bar{r}/w$  para leitura/escrita
- sinais de controle  $clk$  e  $cl$
- sinal de dados de entrada  $data_{in}$  com 8 bits
- sinal de dados de saída  $data_{out}$  com 8 bits

## ■ Registrador FLAGS

- registrador com sinal de carga para palavras de 2 bits
- sinais de controle e dados iguais ao Registrador AC



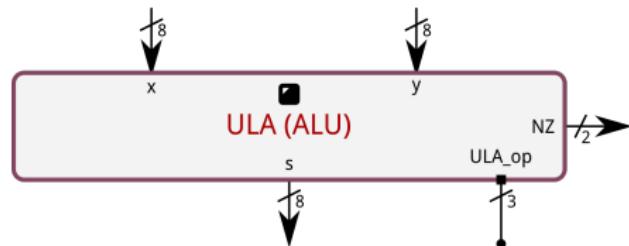
## ■ Observação:

- FLAGS sempre é carregado quando AC é carregado
- sinal de controle  $\bar{r}/w$  é compartilhado entre AC e FLAGS

# Módulo ULA - Componentes II

## ■ ULA (ALU)

- componente que executa as operações Lógicas e operação Aritmética
- entradas:
  - dados  $x$  e  $y$  com 8 bits
  - controle  $ULA\_op$  com 3 bits que determinam operação
- saídas:
  - dados  $s$  com 8 bits e estados  $NZ$  com 2 bits



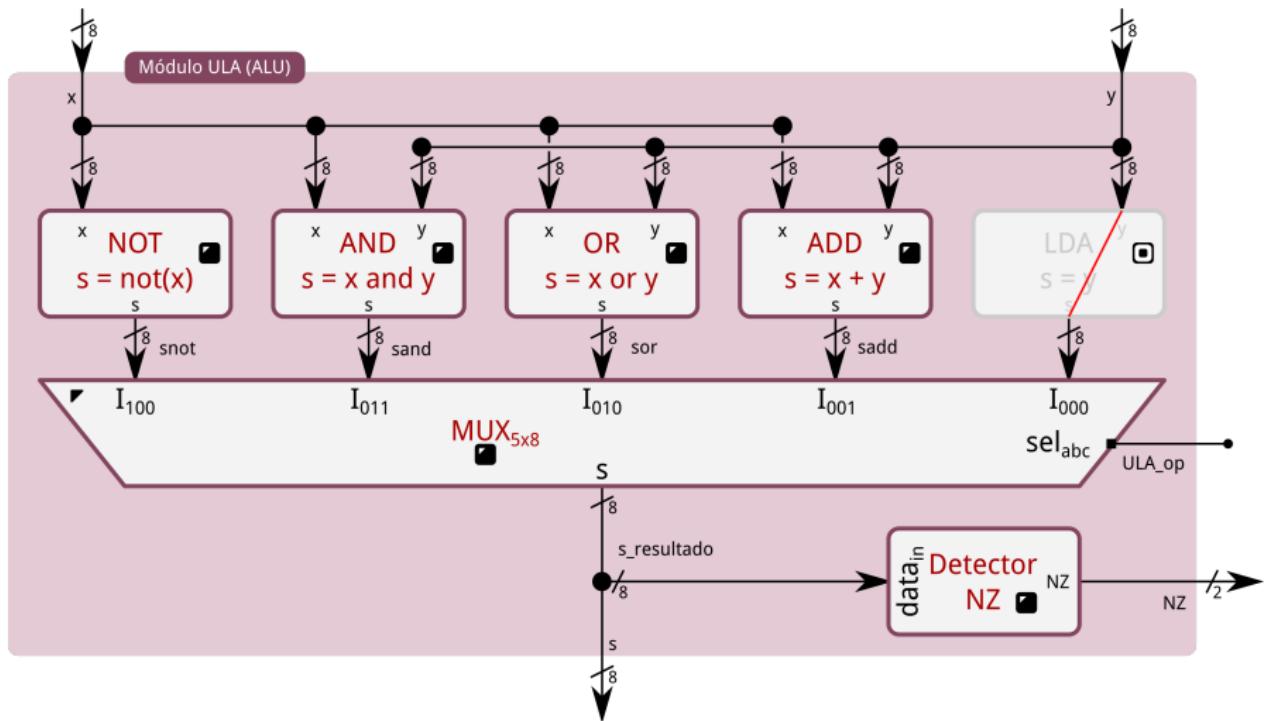
# Módulo ULA - ULA (ALU)

## ■ *ULA\_op*

- determinam operação de acordo com seguinte tabela:

ULA_op ABC	Instrução	Descrição	Operação NEANDER	Descrição ULA (ALU)
000	LDA	AC $\leftarrow$ MEM[end]		$s \leftarrow y$
001	ADD	AC $\leftarrow$ AC + MEM[end]		$s \leftarrow x + y$
010	OR	AC $\leftarrow$ AC or MEM[end]		$s \leftarrow x \text{ or } y$
011	AND	AC $\leftarrow$ AC and MEM[end]		$s \leftarrow x \text{ and } y$
100	NOT	AC $\leftarrow$ not(AC)		$s \leftarrow \text{not}(x)$

# Módulo ULA - ULA (ALU)



# Módulo ULA - ULA (ALU)

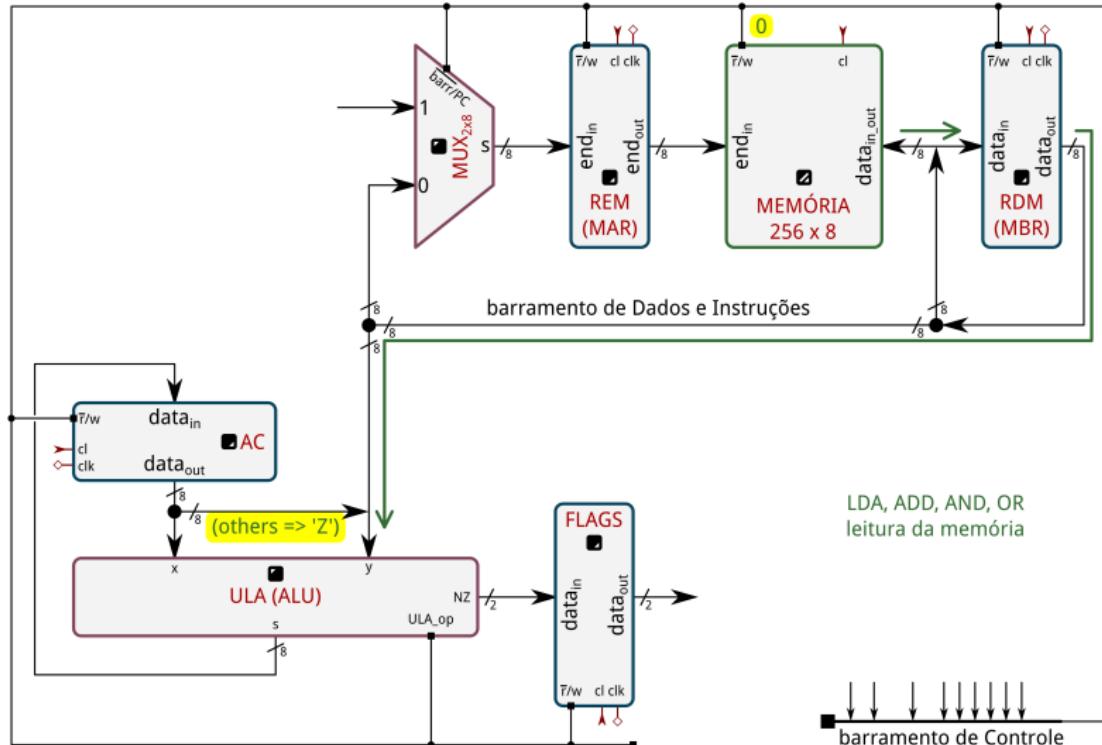
E a armadilha *inout*?

- Em VHDL, uma interface *inout* é uma “porta” bidirecional
  - Funciona tanto como entrada e saída
- Requer cuidados extras
  - Duas entidades escrevendo no mesmo sinal pode causar 'X'
  - Uso de um componente extra  
mantém *inout* em (*others => Z'*) quando a direção é de entrada  
altera *inout* para palavra desejada quando a direção é de saída
  - O outro ‘lado’ (entidade conectada ao sinal) também deve operar da mesma forma

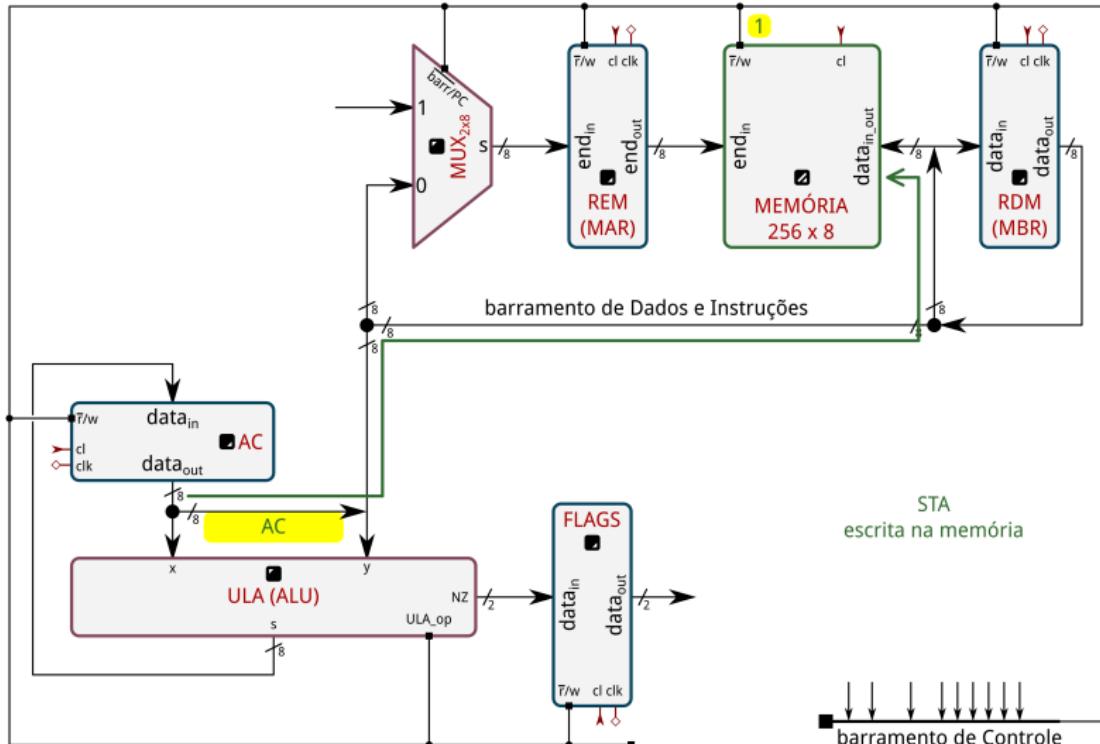
E como se encaixa no NEANDER?

- Existem dois fluxos de dados entre Memória e ULA
  - direção *MEM → ULA*: instruções *LDA*, *ADD*, *AND* e *OR*
  - direção *ULA → MEM*: instrução *STA*

# Módulo ULA - ULA (ALU) - Fluxo *MEM* → *ULA*



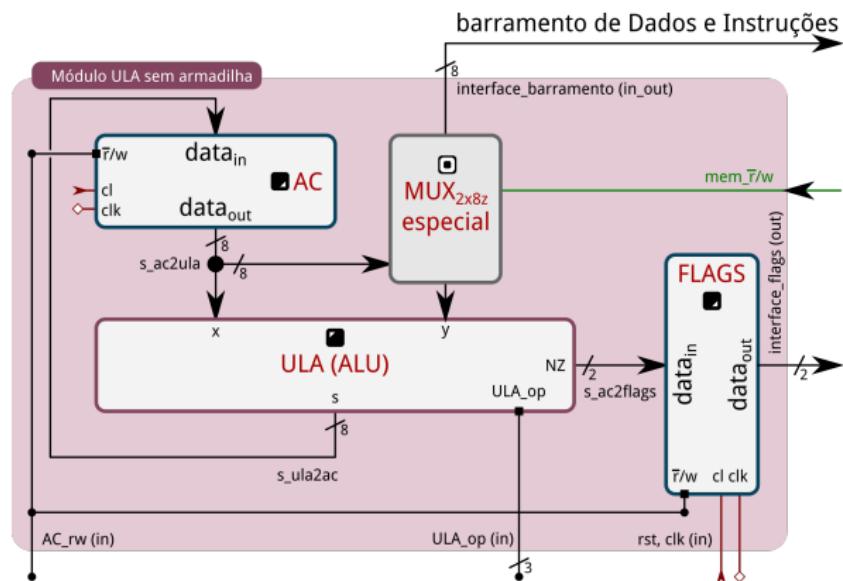
# Módulo ULA - ULA (ALU) - Fluxo $ULA \rightarrow MEM$



# Módulo ULA - ULA (ALU) - resolvendo *inout*

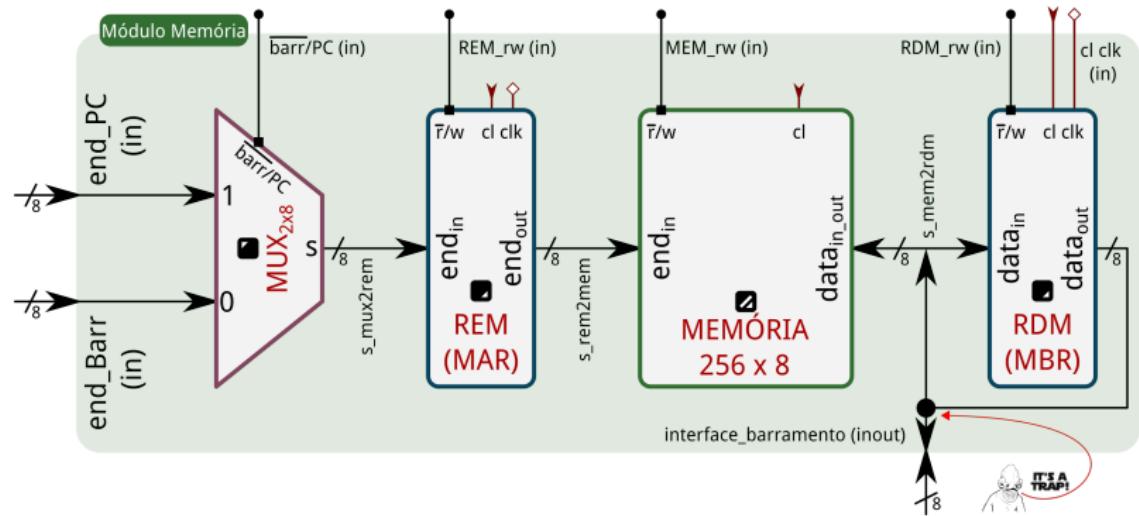
*Multiplexador*<sub>2x8z</sub>

```
barramento <= s_ac2ula when nrw='1' else (others => 'Z');
```



# Organização NEANDER - Memória

# Módulo Memória



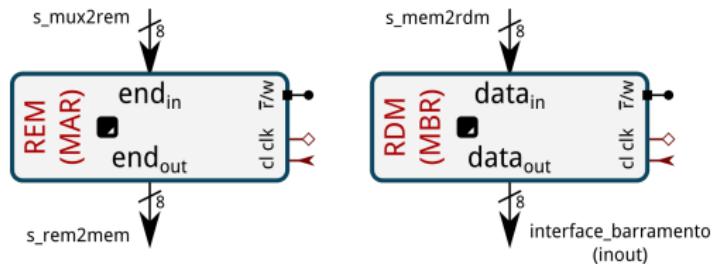
# Módulo Memória - Componentes I

## ■ Registrador *RDM(MBR)*

- sinais de controle e dados semelhantes ao Registrador AC
- armazena temporariamente os dados de saída da Memória
- 8 bits de dados

## ■ Registrador *REM(MAR)*

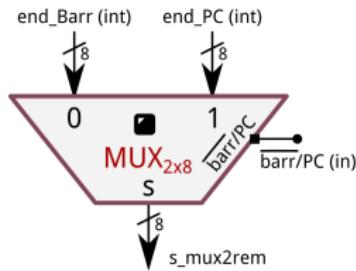
- sinais de controle e dados semelhantes ao Registrador AC
- armazena temporariamente o endereço a ser acessado pela Memória
- 8 bits de endereço



# Módulo Memória - Componentes II

## ■ Multiplexador<sub>2x8</sub>

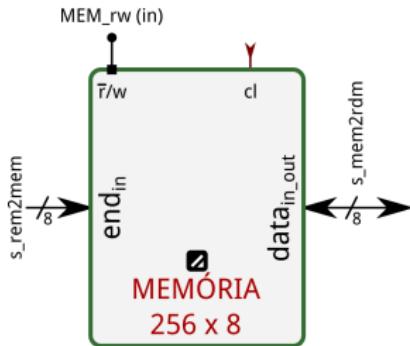
- 2 canais de 8 bits cada um
- Fluxo de endereço do *PC* ou do *Barramento* para *REM(MAR)*
  - Operação :  $REM(MAR) \leftarrow PC$
  - Operação acesso à dados:  $REM(MAR) \leftarrow Barramento$



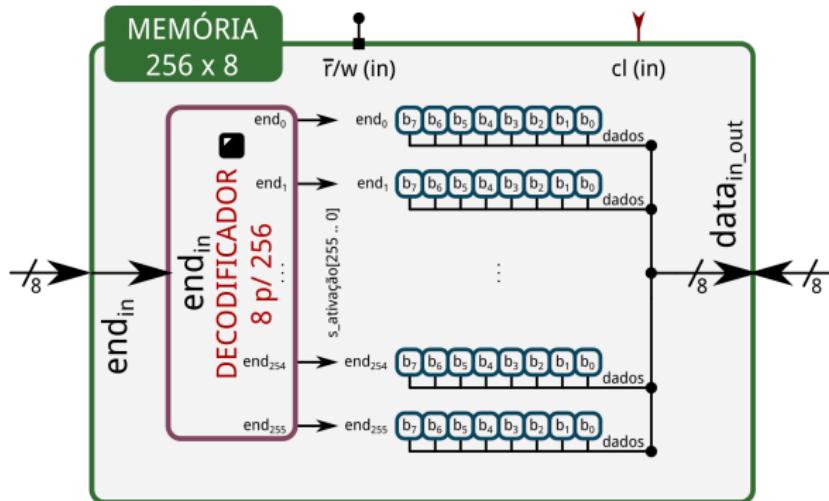
# Módulo Memória - Componentes III

## ■ Memória 256x8

- 256 palavras com 8 bits cada uma
- Uma palavra é selecionada via  $end_{in}$
- Modo de Leitura  $\rightarrow MEM_{rw} = 0$   
 $data_{out}$  retorna palavra armazenada na posição  $end_{in}$
- Modo de Escrita  $\rightarrow MEM_{rw} = 1$   
armazena  $data_{out}$  na posição  $end_{in}$

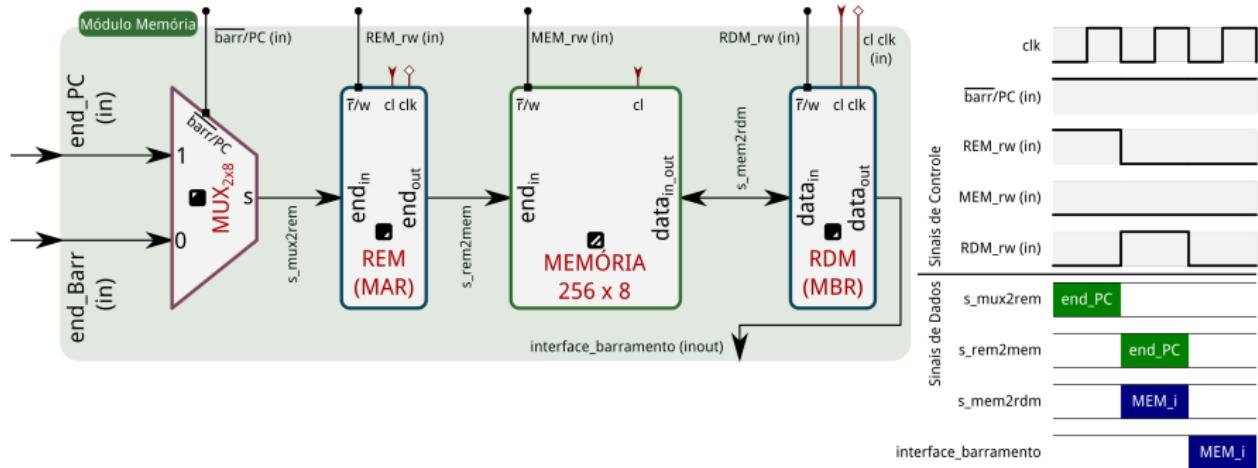


# Módulo Memória - Memória 256x8 - interno

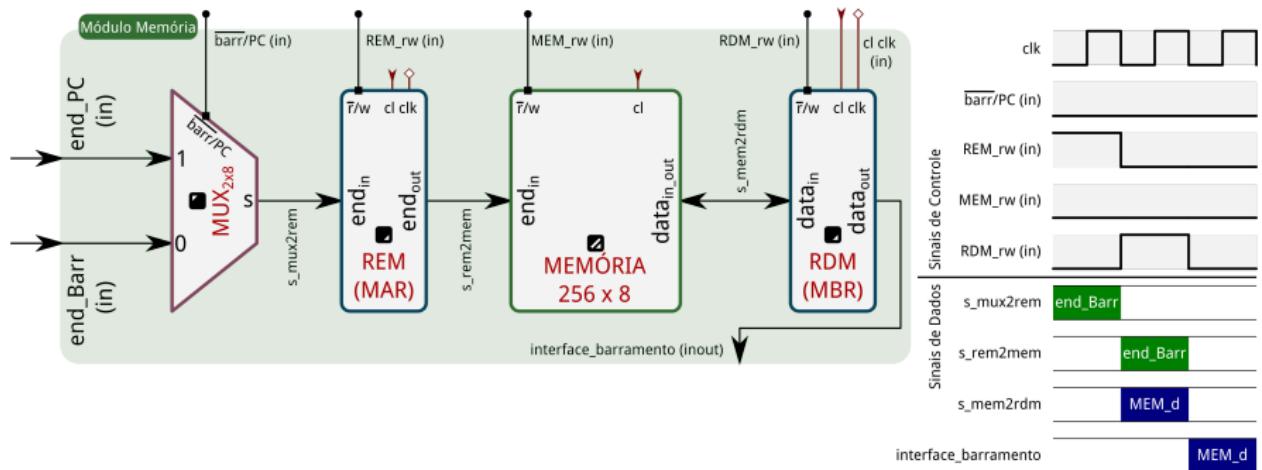


- Não é necessário implementar
  - Verificar pacote NEANDER-Memoria.zip em anexo.

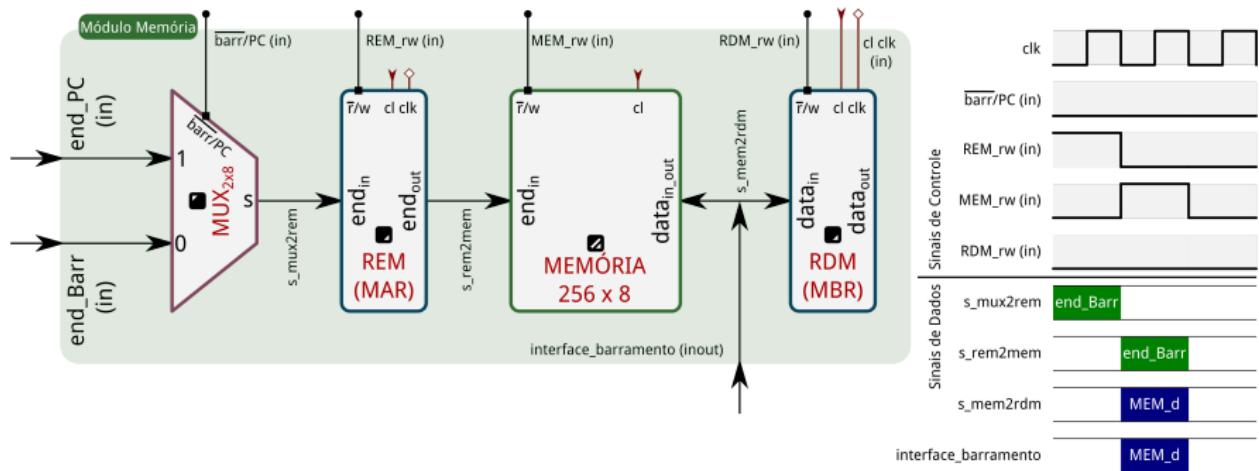
# Módulo Memória - Funcionamento - Leitura de Instrução



# Módulo Memória - Funcionamento - Leitura de Dado



# Módulo Memória - Funcionamento - Gravação de Dado



# Módulo Memória - Funcionamento - Observações

## ■ Ciclos de Clock

- Os ciclos apresentados nos slides anteriores se referem aos tempos locais do Módulo de Memória
- Ao adicionarmos outros Módulos (ULA e UC), os tempos serão alterados. Pois as informações devem estar preparadas antes de serem executadas neste Módulo.

## ■ E a interface *inout*?

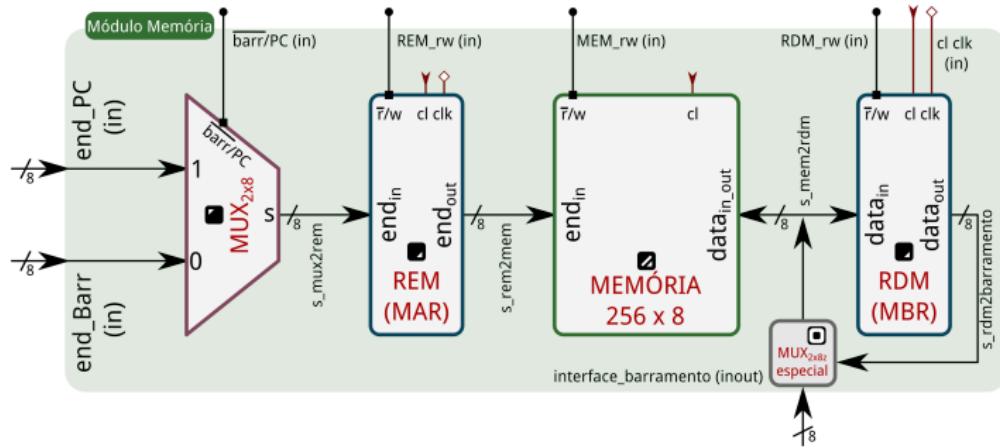
- Resolvida com um truque semelhante ao da *ULA(ALU)*

# Módulo MEM - resolvendo *inout*

*Multiplexador*<sub>2x8z</sub>

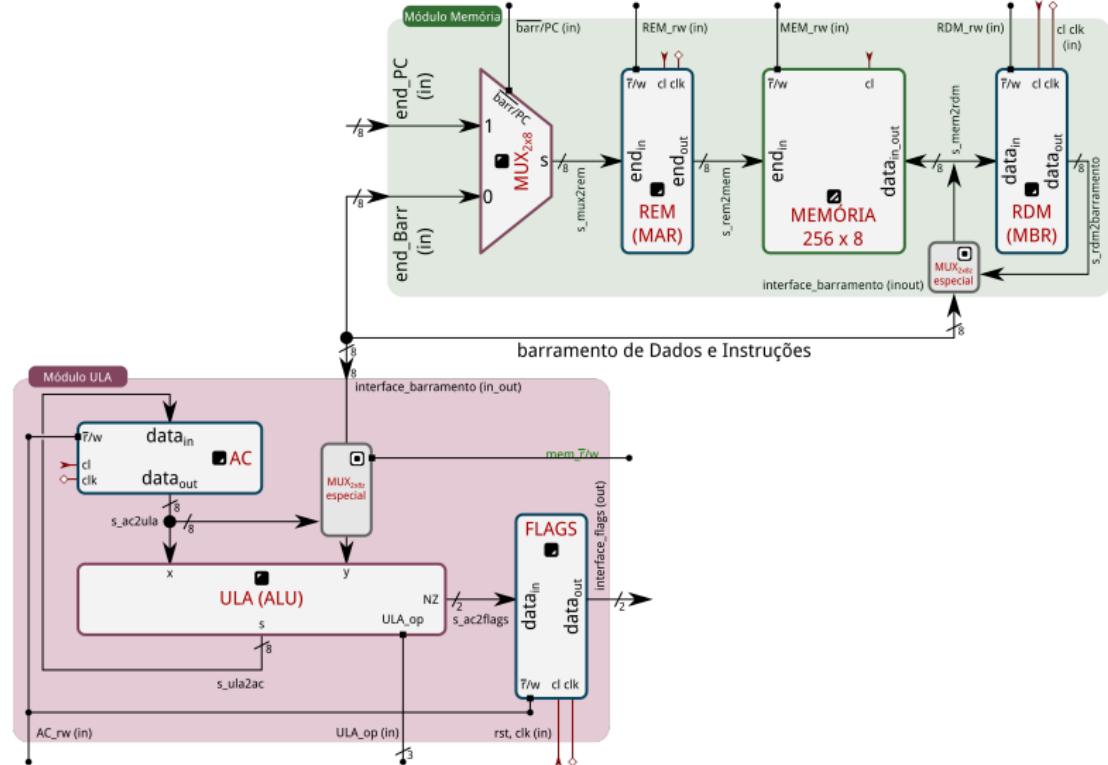
```
barramento <= s_rdm2barr when MEM_nrw = '0'
                           else (others => 'Z');

s_mem2rdm <= barramento when MEM_nrw = '1'
                           else (others => 'Z');
```



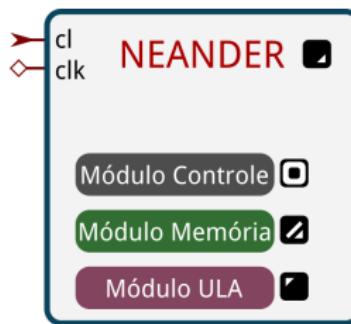
## Organização NEANDER - ULA (ALU) vs MEM

# Módulos ULA(ALU) vs Módulo MEM - Conexão



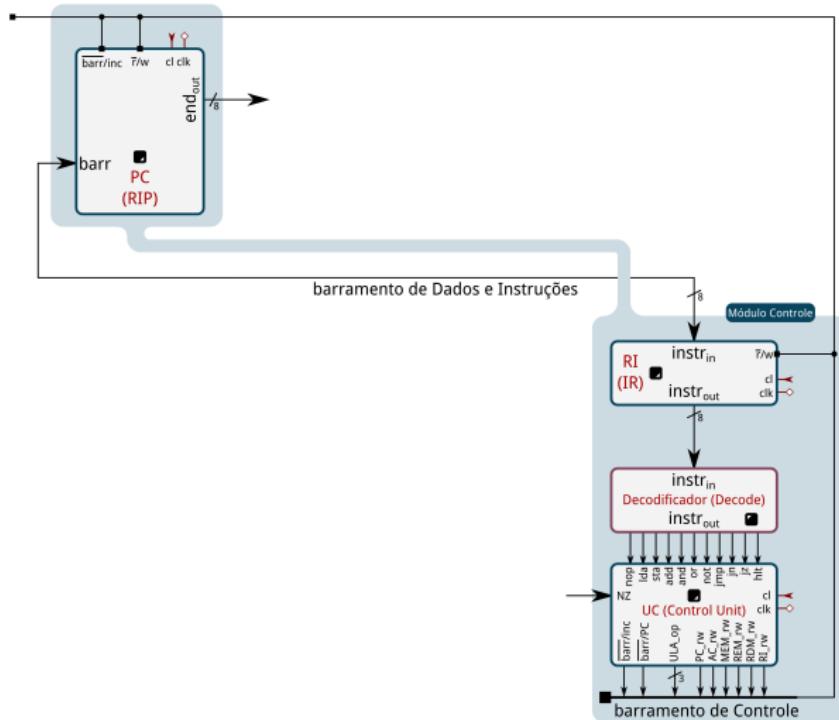
# NEANDER!?

- Já é possível construir a entidade NEANDER
  - Contém os Módulos ULA (ALU) e Memória
  - Interface de entrada: somente *rst* e *clk*



# Organização NEANDER - CONTROLE

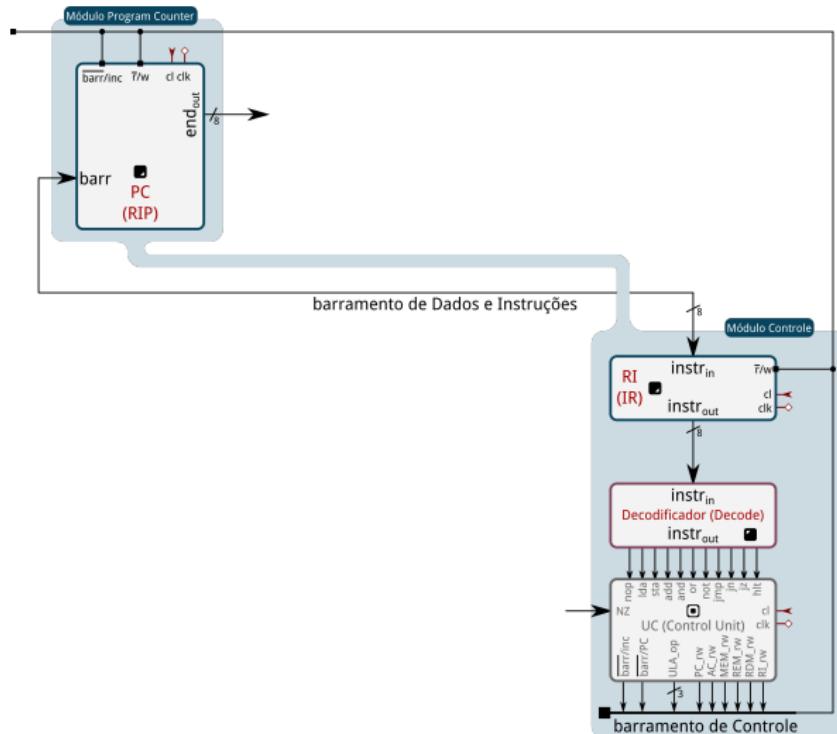
# Módulo Controle



# Módulo Controle - Componentes I

- Program Counter PC (RIP)
  - Incrementador
  - Mux<sub>2x8</sub>
  - Registrador PC
- Registrador RI (IR)
- Decodificador 8 para 11
- Unidade de Controle
  - Contador incremental 0-7
  - Circuitos Combinacionais para cada instrução
  - Gargantua Mux<sub>11x11</sub>
- Dividido em 2 etapas:
  - 1<sup>a</sup>: PC + RI + DEC
  - 2<sup>a</sup>: Unidade de Controle

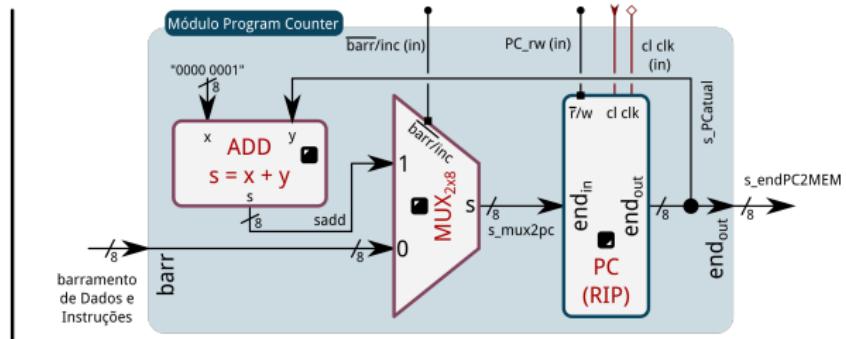
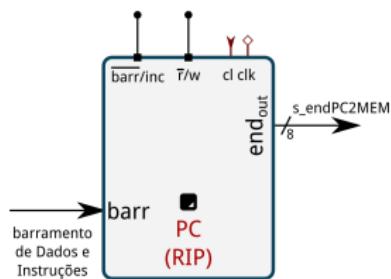
# Módulo Controle - Etapa 01



# Módulo Controle - Etapa 01 - Program Counter (PC/RIP)

- Program Counter PC (RIP)
  - Incrementador
  - Mux<sub>2x8</sub>
  - Registrador PC
- Objetivo:
  - “apontar” para a próxima instrução à ser executada
    - Incremento natural
    - Alteração via instruções JMP, JN ou JZ (salto)

# Módulo Controle - Etapa 01 - Program Counter (PC/RIP)



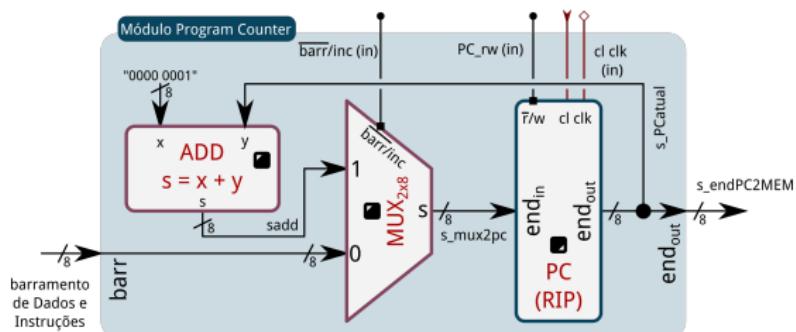
## Sinais de Controle

- tradicionais *cl* e *clk*
- *barr/inc<sub>in</sub>*
  - escolha entre incremento ou salto
- *PC̄/w<sub>in</sub>*
  - execução do incremento ou salto

# Módulo Controle - Etapa 01 - Program Counter (PC/RIP)

## Sinais de Controle

Sinais de Controle		Descrição
barr/inc (in)	PC_rw (in)	
0	0	Não carrega PC
0	1	PC ← barr   para saltos
1	0	Não carrega PC
1	1	PC ← PC++   incremento



# Módulo Controle - Etapa 01 - RI e Decodificador

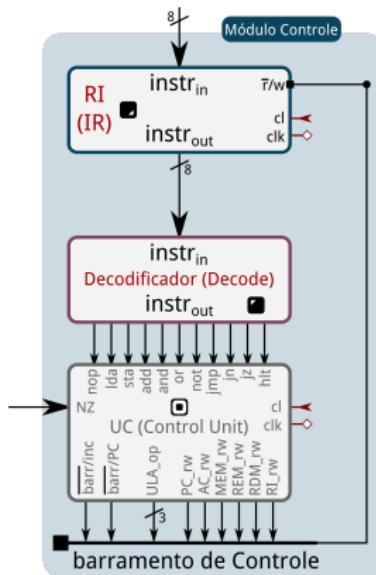
## ■ Registrador de Instruções

- sinais de controle e “dados” semelhantes ao Registrador AC
- armazena instrução para execução
  - Tempo de vida da instrução

## ■ Decodificador 8 para 11

- Identifica a instrução à ser executada
- Informa à U.C.

# Módulo Controle - Etapa 01 - RI e Decodificador



# Módulo Controle - Etapa 01 - Decodificador

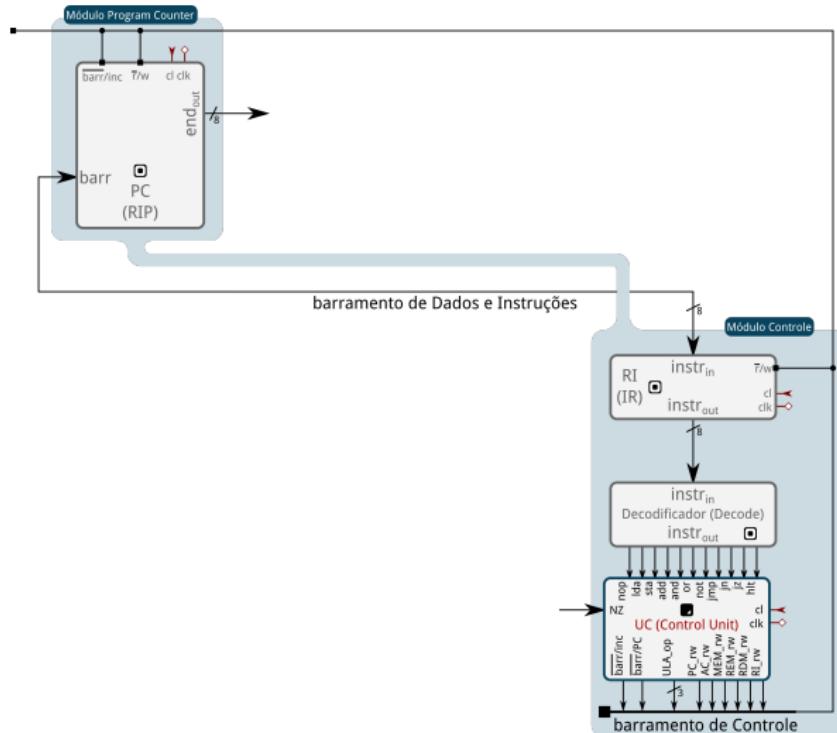
## Decodificador 8 para 11

- Identifica a instrução à ser executada
- Informa à U.C.

Instr.	Entrada de RI			Saídas									
	Opcode	dec		snop	ssta	slda	sadd	sor	sand	snot	sjmp	sjn	sjz
NOP	0000 0000	0		1									
STA	0001 0000	16			1								
LDA	0010 0000	32				1							
ADD	0011 0000	48					1						
OR	0100 0000	64						1					
AND	0101 0000	80							1				
NOT	0110 0000	96								1			
JMP	1000 0000	128									1		
JN	1001 0000	144										1	
JZ	1010 0000	160										1	
HLT	1111 0000	240											1

\* células em branco = '0'

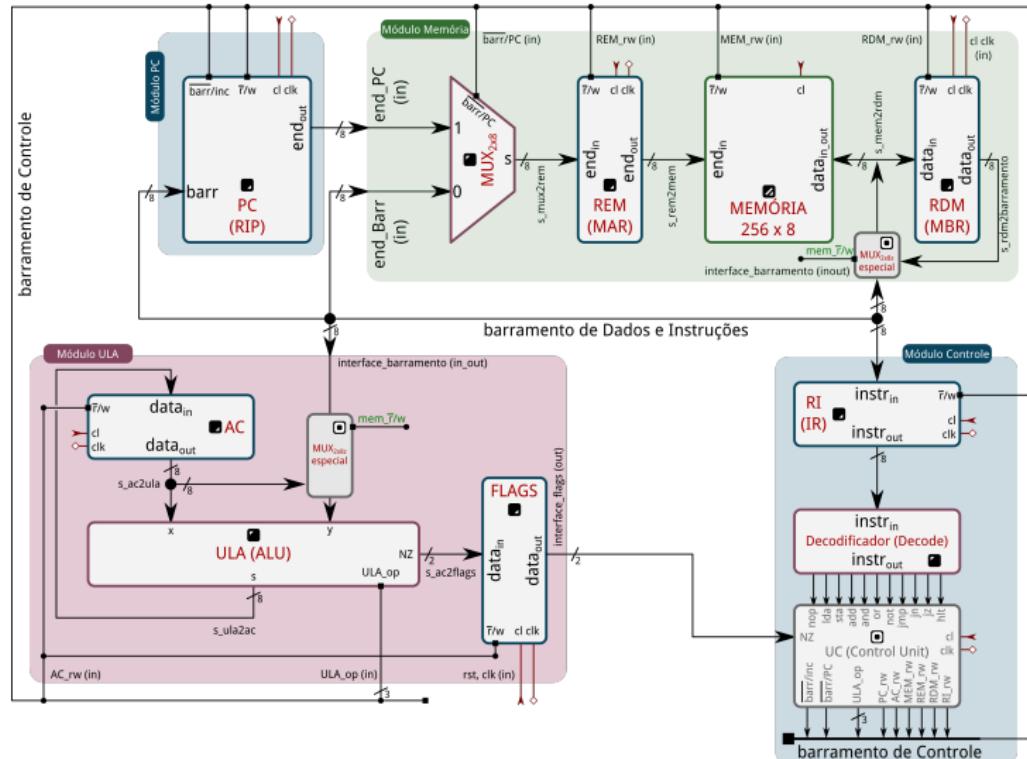
# Módulo Controle - Etapa 02



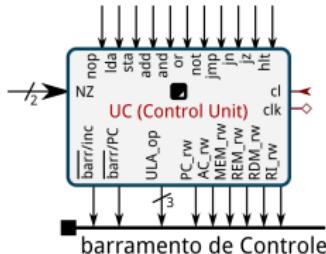
# Módulo Controle - Etapa 02 - Mas antes ...

- Conectar todos os módulos anteriores
  - Módulo ULA
  - Módulo Memória
  - Módulo Controle - Program Counter (PC)
- Sem os módulos conectados, não é possível compreender o que fará a seguir!

# Módulo Controle - Etapa 02 - Mas antes ...



# Módulo Controle - Etapa 02 - Unidade de Controle



## ■ Componentes:

- Contador Síncrono incremental 0-7
- Circuitos Combinacionais para cada instrução
- Mux<sub>11x11</sub>

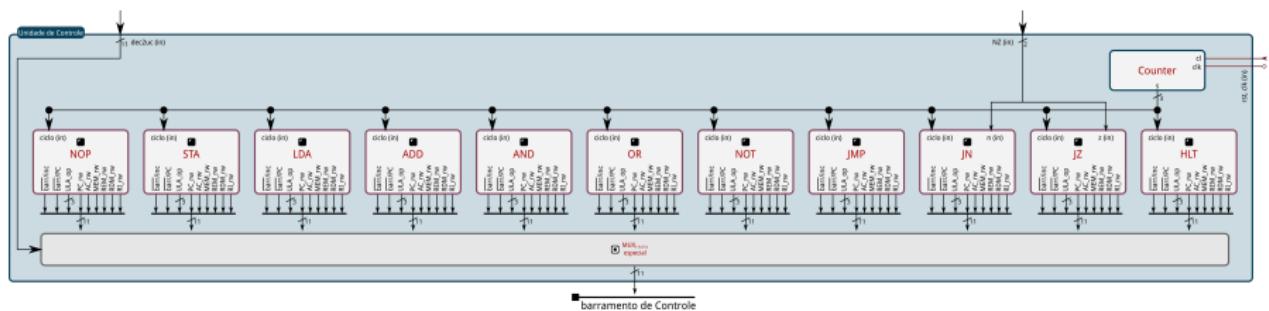
## ■ Função

- Acionar sinais de controle no tempo correto para executar as instruções NEANDER
- Executar Ciclo de Instrução

## ■ Interface:

- Entradas: seletores de instruções, comuns *cl* e *clk*, e *NZ*
- Saídas: sinais de controle para componentes e caminhos

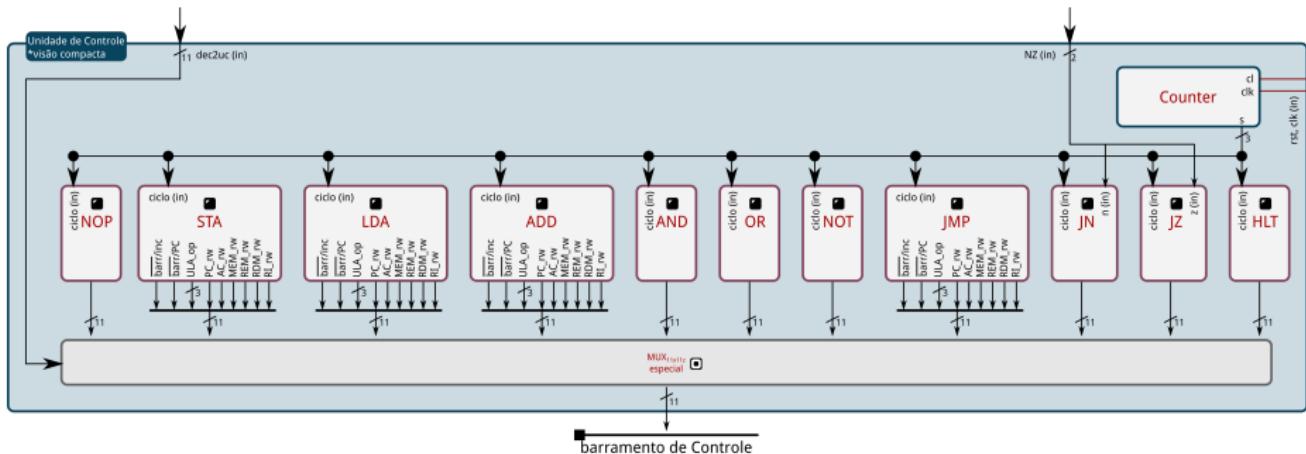
# Módulo Controle - Etapa 02 - Unidade de Controle



wait for it!

# Módulo Controle - Etapa 02 - Unidade de Controle

Visão compacta



# Módulo Controle - Conceitos

## ■ Execução de um Programa

- Sequência finita e ordenada de ações (instruções) com objetivo de solucionar um problema

## ■ Instrução

- É uma ação executada por um processador  
Exemplo: ADD, LDA, STA

## ■ Ciclo de Instrução

- Tempo necessário para executar completamente uma Instrução  
Dividido em vários subciclos (dependente de implementação de máquina)

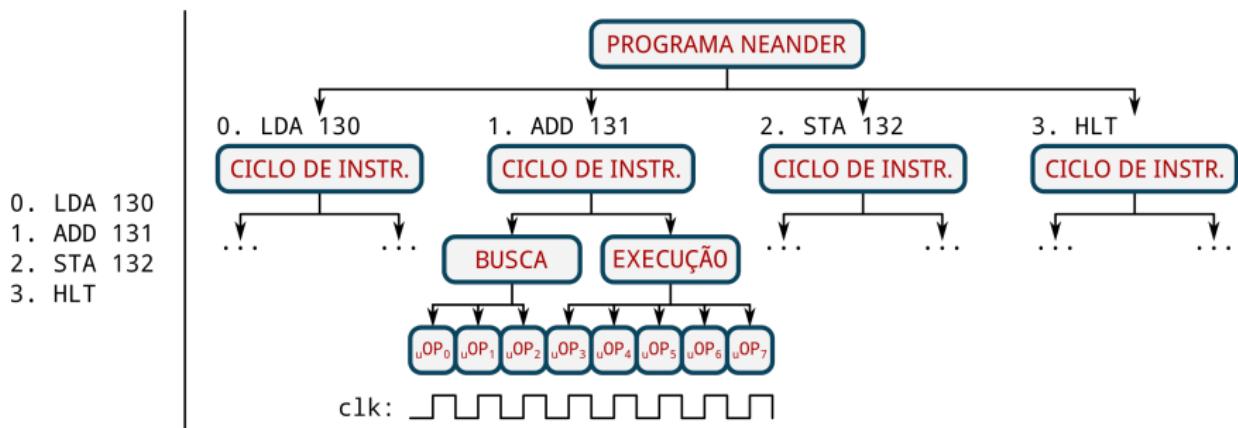
Mais comuns: Ciclo de Busca e Ciclo de Execução  
Composto por várias Micro-operação  $\mu op$

## ■ Micro-operação $\mu op$

- É um “passo atômico” que altera algum fluxo de informação durante um ciclo de clock

# Módulo Controle - Conceitos

- Relação entre Programa, Ciclo de Instrução, subciclos e  $\mu$ op



# Módulo Controle - Ciclo de Instrução NEANDER

## ■ Ciclo de Busca

- PC aponta para próxima instrução
- RI deve receber instrução da memória apontada por PC
- Incrementar PC

## ■ Ciclo de Execução

- Depende da instrução a ser executada
- Aritmética e Lógica
  - PC aponta para operando
  - AC deve receber AC  $op\_ULA$  memória
  - Incrementar PC
- Salto e Salto condicional verdadeiro
  - PC aponta para endereço de destino
  - PC deve receber memória

# Módulo Controle - Exemplo: Ciclo de Instrução LDA

## ■ Ciclo de Busca

- PC aponta para próxima instrução
- RI deve receber instrução da memória apontada por PC
- Incrementar PC

## ■ Ciclo de Execução

- Aritmética e Lógica
  - PC aponta para operando
  - AC deve receber memória
  - Incrementar PC

# Módulo Controle - Exemplo: Ciclo de Instrução LDA

LDA: ( $ULA\_op = 000$ )

## Ciclo de BUSCA

0 : REM  $\leftarrow$  PC  
1 : RDM  $\leftarrow$  MEM  
PC++  
2 : RI  $\leftarrow$  RDM

## Ciclo de EXECUÇÃO

3 : REM  $\leftarrow$  PC  
4 : RDM  $\leftarrow$  MEM  
PC++  
5 : REM  $\leftarrow$  Barr  
6 : RDM  $\leftarrow$  MEM  
7 : AC  $\leftarrow$  RDM

# Módulo Controle - Exemplo: Ciclo de Instrução LDA

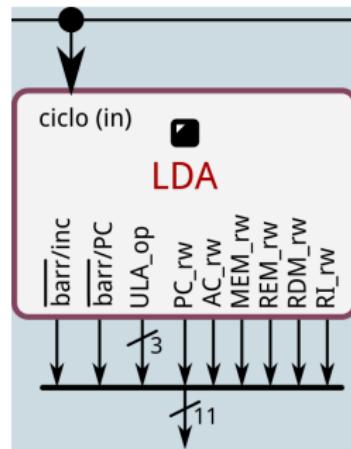
Ciclo	LDA		Saídas (sinais de controle)								
	Contador b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	dec	barr/inc	barr/PC	ULA_op	PC_́rw	AC_́rw	MEM_́rw	REM_́rw	RDM_́rw	RI_́rw
Busca	000	0	1	1	000	0	0	0	1	0	0
	001	1	1	1	000	1	0	0	0	1	0
	010	2	1	1	000	0	0	0	0	0	1
Exec	011	3	1	1	000	0	0	0	1	0	0
	100	4	1	1	000	1	0	0	0	1	0
	101	5	1	0	000	0	0	0	1	0	0
	110	6	1	1	000	0	0	0	0	1	0
	111	7	1	1	000	0	1	0	0	0	0

# Módulo Controle - Exemplo: Ciclo de Instrução LDA

Expressões booleanas dos sinais de controle para Ciclo de Instrução LDA:

- $\overline{barr}/inc = 1$
- $\overline{barr}/PC = \overline{b}_2 \vee b_1 \vee \overline{b}_0$
- $ULA\_op = 000$
- $PC\_rw = \overline{b}_1 \wedge (b_2 \oplus b_0)$
- $AC\_rw = b_2 \wedge b_1 \wedge b_0$
- $MEM\_rw = 0$
- $REM\_rw = (\overline{b}_1 \wedge (b_2 \odot b_0)) \vee (\overline{b}_2 \wedge b_1 \wedge b_0)$
- $RDM\_rw = (b_2 \wedge \overline{b}_0) \vee (\overline{b}_2 \wedge \overline{b}_1 \wedge b_0)$
- $RI\_rw = \overline{b}_2 \wedge b_1 \wedge \overline{b}_0$

# Módulo Controle - Módulo LDA



# Módulo Controle - Ciclo de Instrução: NOP e STA

NOP: (ULA<sub>op</sub> = 000)

## Ciclo de BUSCA

0 : REM  $\leftarrow$  PC  
1 : RDM  $\leftarrow$  MEM  
PC++  
2 : RI  $\leftarrow$  RDM

STA: (ULA<sub>op</sub> = 000)

## Ciclo de BUSCA

0 : REM  $\leftarrow$  PC  
1 : RDM  $\leftarrow$  MEM  
PC++  
2 : RI  $\leftarrow$  RDM

## Ciclo de EXECUÇÃO

3 :  
4 :  
5 :  
6 :  
7 :

## Ciclo de EXECUÇÃO

3 : REM  $\leftarrow$  PC  
4 : RDM  $\leftarrow$  MEM  
PC++  
5 : REM  $\leftarrow$  Barr  
6 : MEM  $\leftarrow$  AC  
7 :

# Módulo Controle - Ciclo de Instrução: NOT

NOT: ( $ULA\_op = 100$ )

## Ciclo de BUSCA

```
0 : REM ← PC  
1 : RDM ← MEM  
     PC++  
2 : RI   ← RDM
```

## Ciclo de EXECUÇÃO

```
3 :  
4 :  
5 :  
6 :  
7 : AC   ← RDM (AC ← s_ula2ac)
```

# Módulo Controle - Ciclo de Instrução: ADD, OR e AND

ADD: ( $ULA\_op = 001$ )

Ciclo de BUSCA

- 0 :  $REM \leftarrow PC$
- 1 :  $RDM \leftarrow MEM$
- $PC++$
- 2 :  $RI \leftarrow RDM$

Ciclo de EXECUÇÃO

- 3 :  $REM \leftarrow PC$
- 4 :  $RDM \leftarrow MEM$
- $PC++$
- 5 :  $REM \leftarrow Barr$
- 6 :  $RDM \leftarrow MEM$
- 7 :  $AC \leftarrow RDM$

Semelhante ao LDA

- Altera sinal  $ULA\_op$

OR: ( $ULA\_op = 010$ )

- Altera sinal  $ULA\_op$

AND: ( $ULA\_op = 011$ )

- Altera sinal  $ULA\_op$

# Módulo Controle - Ciclo de Instrução: JMP

JMP: ( $ULA\_op = 000$ )

JN (verdadeiro)

- se  $N = 1$  então JMP

## Ciclo de BUSCA

0 : REM  $\leftarrow$  PC  
1 : RDM  $\leftarrow$  MEM  
PC++  
2 : RI  $\leftarrow$  RDM

JZ (verdadeiro)

- se  $Z = 1$  então JMP

## Ciclo de EXECUÇÃO

3 : REM  $\leftarrow$  PC  
4 : RDM  $\leftarrow$  MEM  
5 : PC  $\leftarrow$  Barr  
6 :  
7 :

# Módulo Controle - Ciclo de Instrução: JN JZ Falso

JN e JZ Falso: ( $ULA\_op = 000$ )

## Ciclo de BUSCA

- 0 :  $REM \leftarrow PC$
- 1 :  $RDM \leftarrow MEM$
- $PC++$
- 2 :  $RI \leftarrow RDM$

JN (falso)

- se  $N = 0$  então  $PC++$

JZ (falso)

- se  $Z = 0$  então  $PC++$

## Ciclo de EXECUÇÃO

- 3 :  $PC++$
- 4 :
- 5 :
- 6 :
- 7 :

# Módulo Controle - Ciclo de Instrução: HLT

HLT: ( $ULA\_op = 000$ )

- Não faz nada!

Ciclo de BUSCA

0 :

1 :

2 :

Ciclo de EXECUÇÃO

3 :

4 :

5 :

6 :

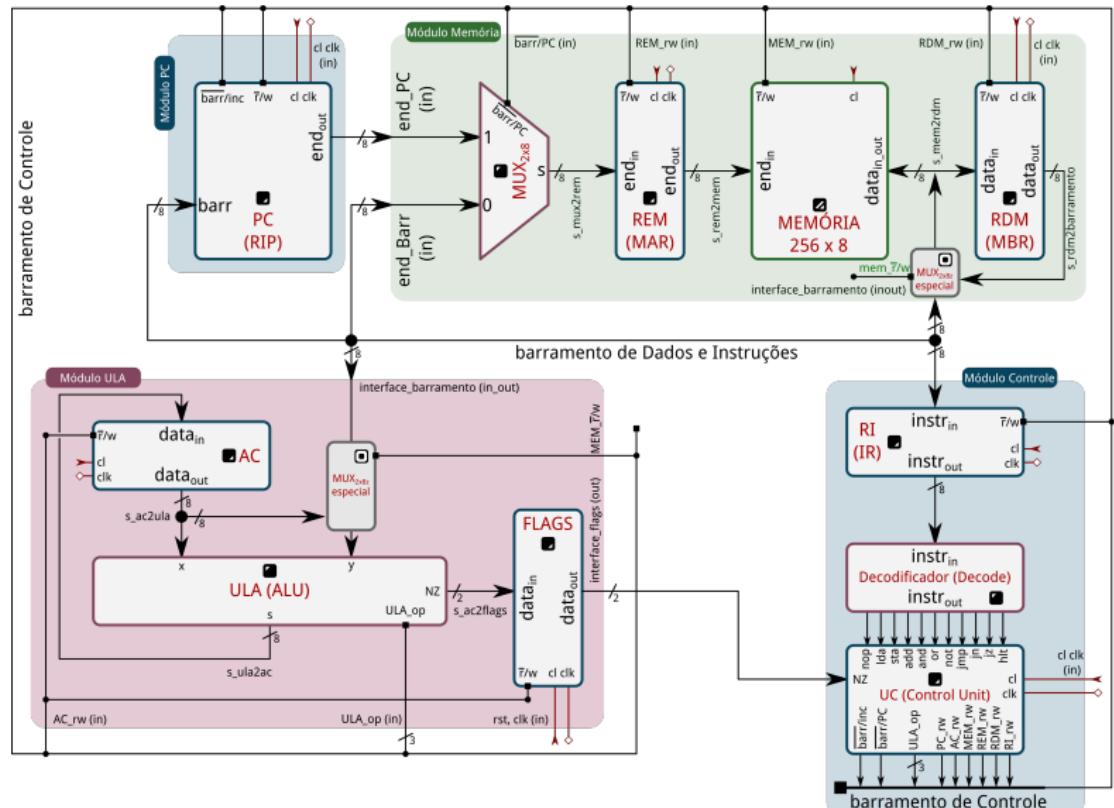
7 :

# Módulo Controle - *Multiplexador*<sub>11x11z</sub>



```
s <= b_nop when sel_op = "10000000000" else
      b_sta when sel_op = "01000000000" else
      ...
      b_hlt when sel_op = "00000000001" else
(others => 'Z');
```

# NEANDER - Final form!



# NEANDER - TESTES

## Teste para NEANDER - VHDL

Simular todos os exercícios indicados no Slide 20



```
belfiorini@B-Note-Mini-EAB: ~/Univates/2020-online/part2/E1_SD_Principal/Waves/Parte04/yVHDL/Projeto_04  
Arquivo Editar Ver Pessoal Terminal Ativa  
NEAMER: $ ghsel -a t_vhdli -i ghsel -r tb_neamer --wavenumber NEAMER_tb2023.ghw --stop-time=1000ns  
NEAMER: $ info: simulation stopped by --stop-time @1us  
NEAMER: $
```

# Atividade avaliativa

- Será especificada em arquivo separado
  - publicado no MSTeams