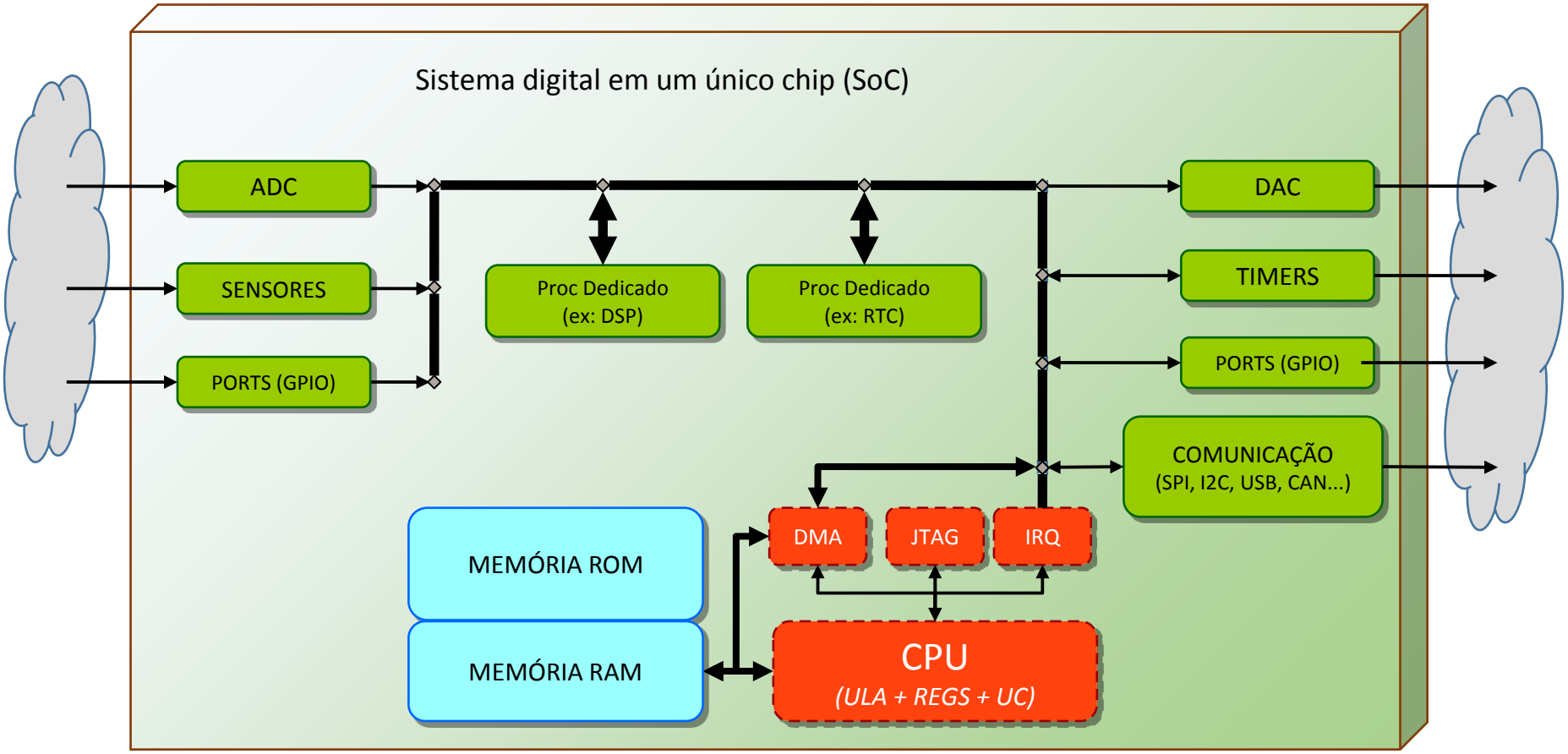
✧ System on Chip (sistemas μ processados)

João Ranhel

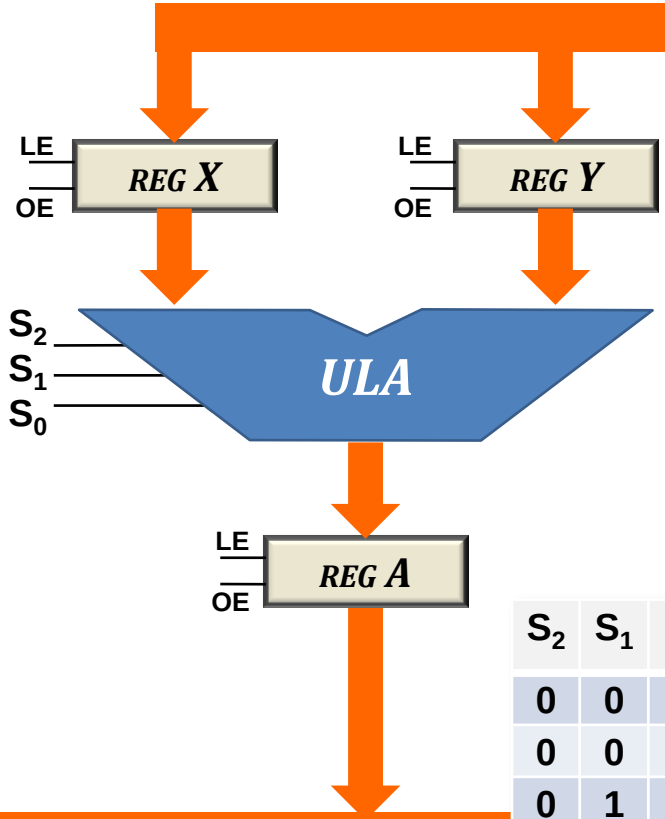


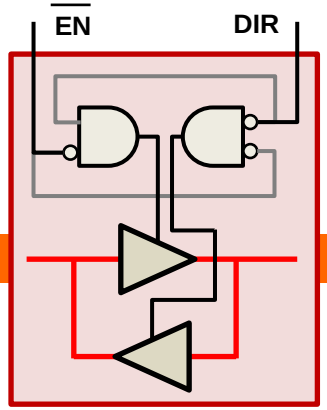
Objetivo da disciplina:

- Compreender o que são e como funcionam sistemas digitais μ processados
- Aprender a utilizar (aplicar) *μP* para solucionar problemas de engenharia
- O enfoque será mais em SoC (*microcontroladores*).

a) Descreva a sequência de ***operações*** p/ calcular: $A = X + Y$

b) Faça a tabela de comandos (no nível lógico) para que esta operação seja executada.





S_2	S_1	S_0	Operação
0	0	0	X AND Y
0	0	1	X OR Y
0	1	0	NOT X
0	1	1	X XOR Y
1	0	0	$X + Y$
1	0	1	$X - Y$
1	1	0	$Y - X$
1	1	1	...

Resposta...

1° passo: carregar ***x***

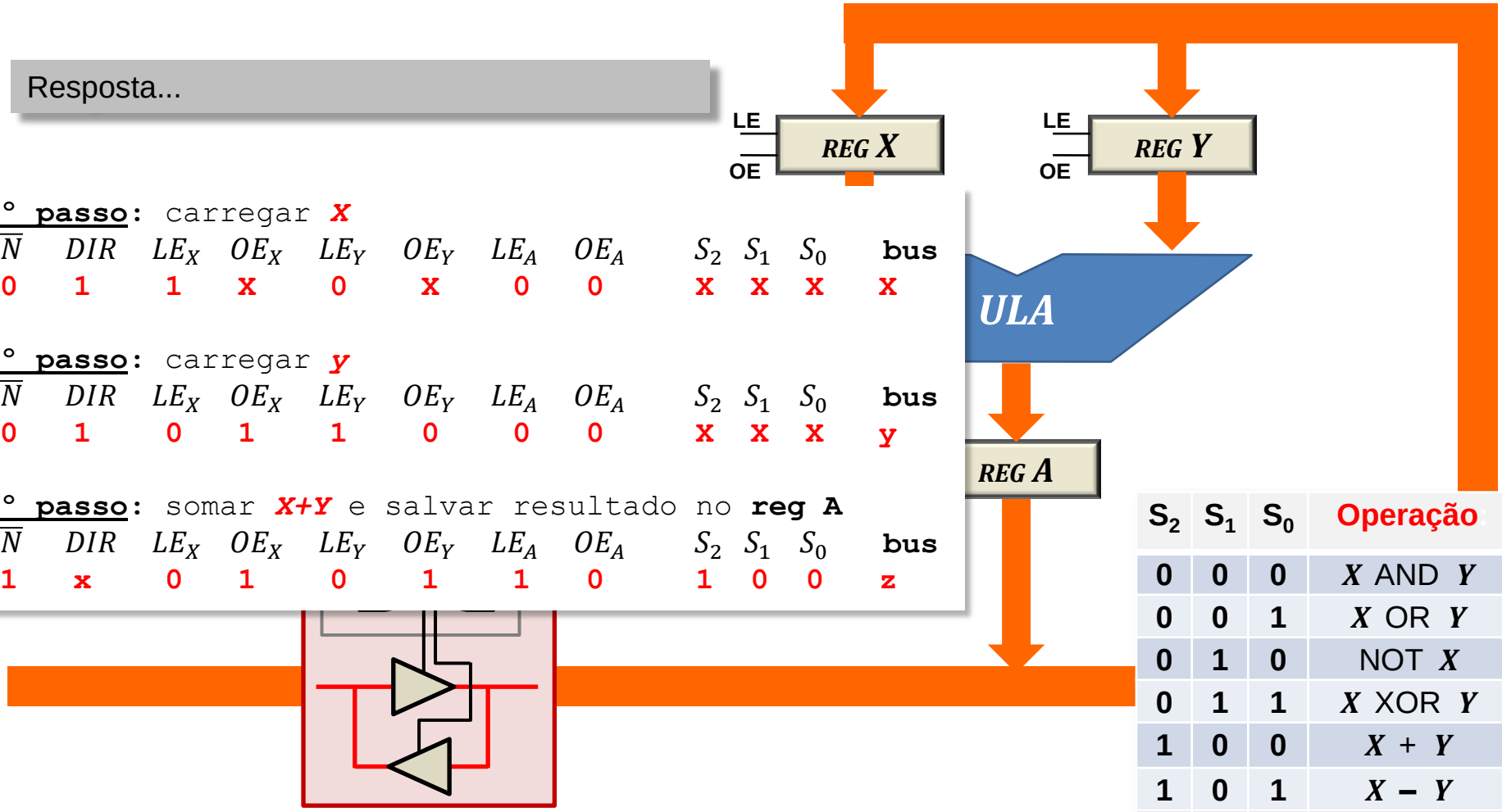
$\overline{EN}$	DIR	LE_X	OE_X	LE_Y	OE_Y	LE_A	OE_A	S_2	S_1	S_0	bus
0	1	1	x	0	x	0	0	x	x	x	x

2° passo: carregar ***y***

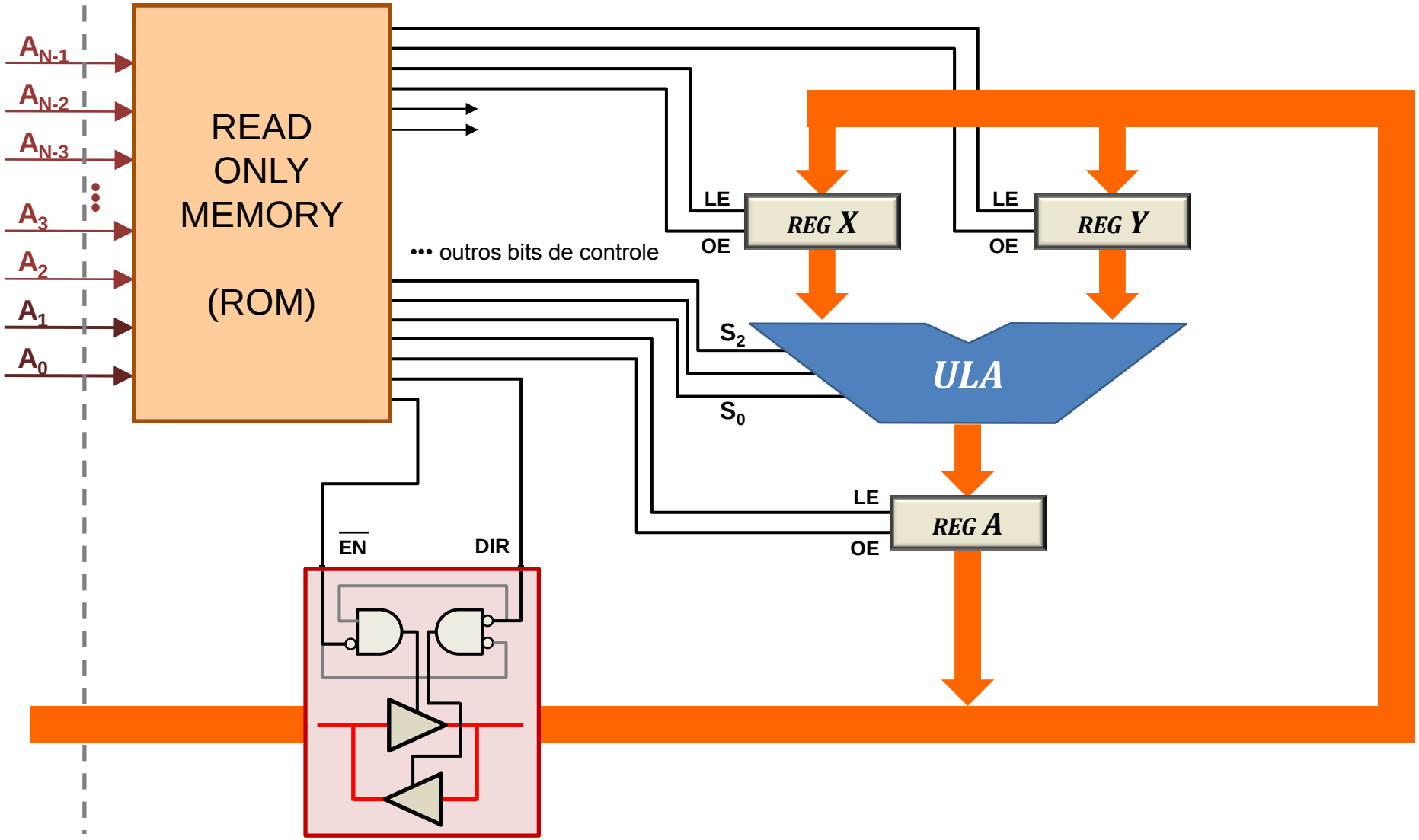
$\overline{EN}$	DIR	LE_X	OE_X	LE_Y	OE_Y	LE_A	OE_A	S_2	S_1	S_0	bus
0	1	0	1	1	0	0	0	x	x	x	y

3° passo: somar ***x+y*** e salvar resultado no **reg A**

$\overline{EN}$	DIR	LE_X	OE_X	LE_Y	OE_Y	LE_A	OE_A	S_2	S_1	S_0	bus
1	x	0	1	0	1	1	0	1	0	0	z



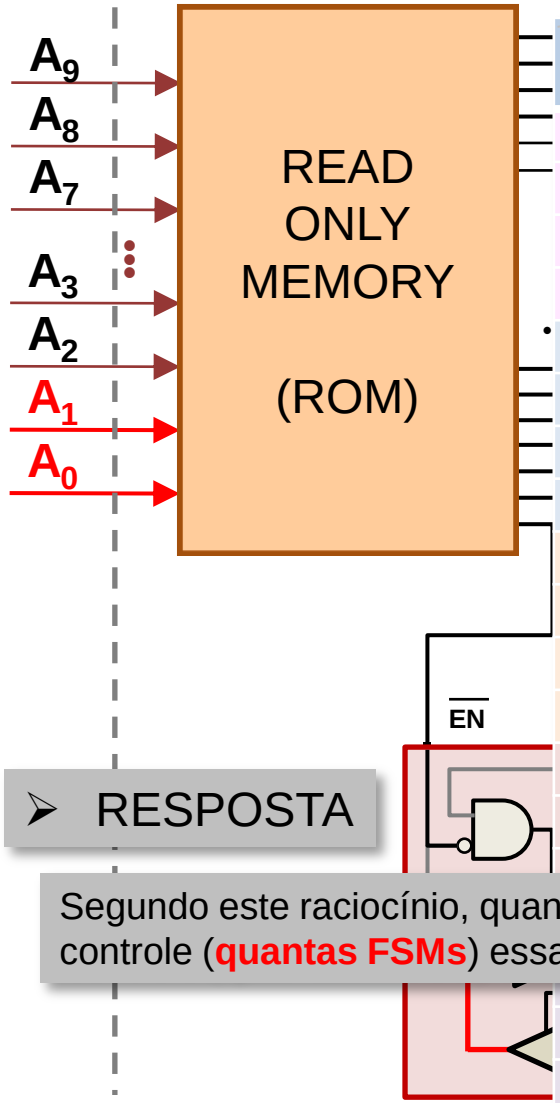
S_2	S_1	S_0	Operação
0	0	0	X AND Y
0	0	1	X OR Y
0	1	0	NOT X
0	1	1	X XOR Y
1	0	0	$X + Y$
1	0	1	$X - Y$
1	1	0	$Y - X$
1	1	1	...



✓ É trabalho do projetista do μ*P* gravar a ***MEMÓRIA*** com o ***FIRMWARE***.

UFABC

✧ AUTOMATIZANDO O CONTROLE da sequência de operações



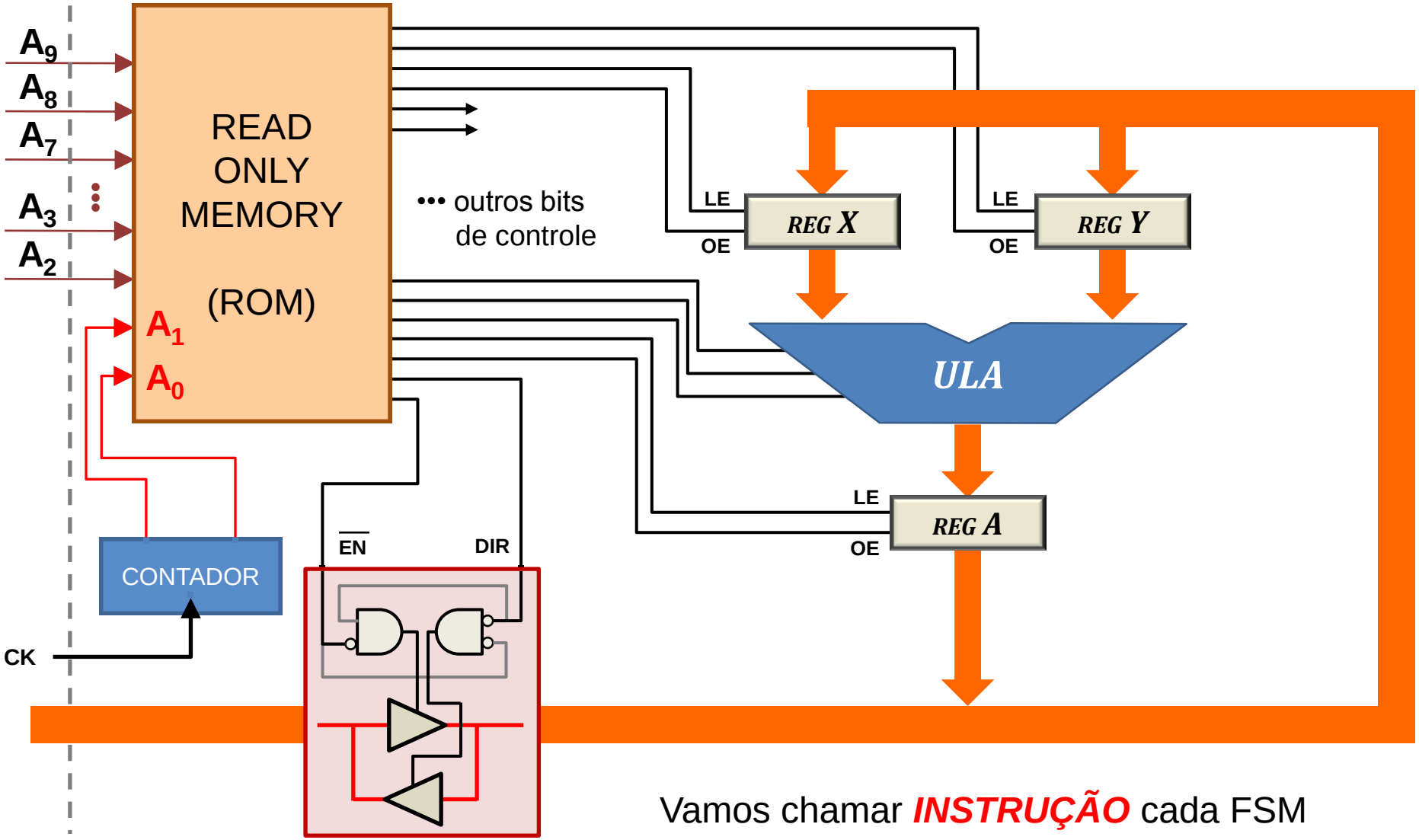
ENDEREÇO BIN	OPERAÇÃO (FSM)	DADO HEX
00 0001 0011	I_004: out A	0x54F2
00 0001 0010	I_004: A←X^Y	0x360F
00 0001 0001	I_004: load Y	0xBC34
00 0001 0000	I_004: load X	0x5286
00 0000 1111		
00 0000 1110		
00 0000 1101		
00 0000 1100		
00 0000 1011	I_002: out A	0x54F2
00 0000 1010	I_002: A←X-Y	0x66D9
00 0000 1001	I_002: load Y	0xBC34
00 0000 1000	I_002: load X	0x5286
00 0000 0111		
00 0000 0110		
	out A	0x58C4
00 0000 0010	I_000: A←X+Y	0x7F02
00 0000 0001	I_000: load Y	0xBC34
00 0000 0000	I_000: load X	0x5286

ENDEREÇO BIN	OPERAÇÃO (FSM)	DADO HEX
11 1111 1111	I_004: out A	0x50D2
11 1111 1110	I_004: A←X!Y	0x360F
11 1111 1101	I_004: load Y	0xBC34
11 1111 1100	I_004: load X	0x5286
...		
...		
...		
...		
11 1111 1011	I_002: out A	0xC311
11 1111 1010	I_002: A←X•Y	0x66D9
11 1111 1001	I_002: load Y	0xBC34
11 1111 1000	I_002: load X	0x5286
11 1101 0111		
11 1101 0110		
11 1101 0101		
11 1101 0100		
11 1100 0011	I_000: out A	0x58C4
11 1100 0010	I_000: A←X|Y	0x7F02
11 1100 0001	I_000: load Y	0xBC34
11 1100 0000	I_000: load X	0x5286

➢ RESPOSTA

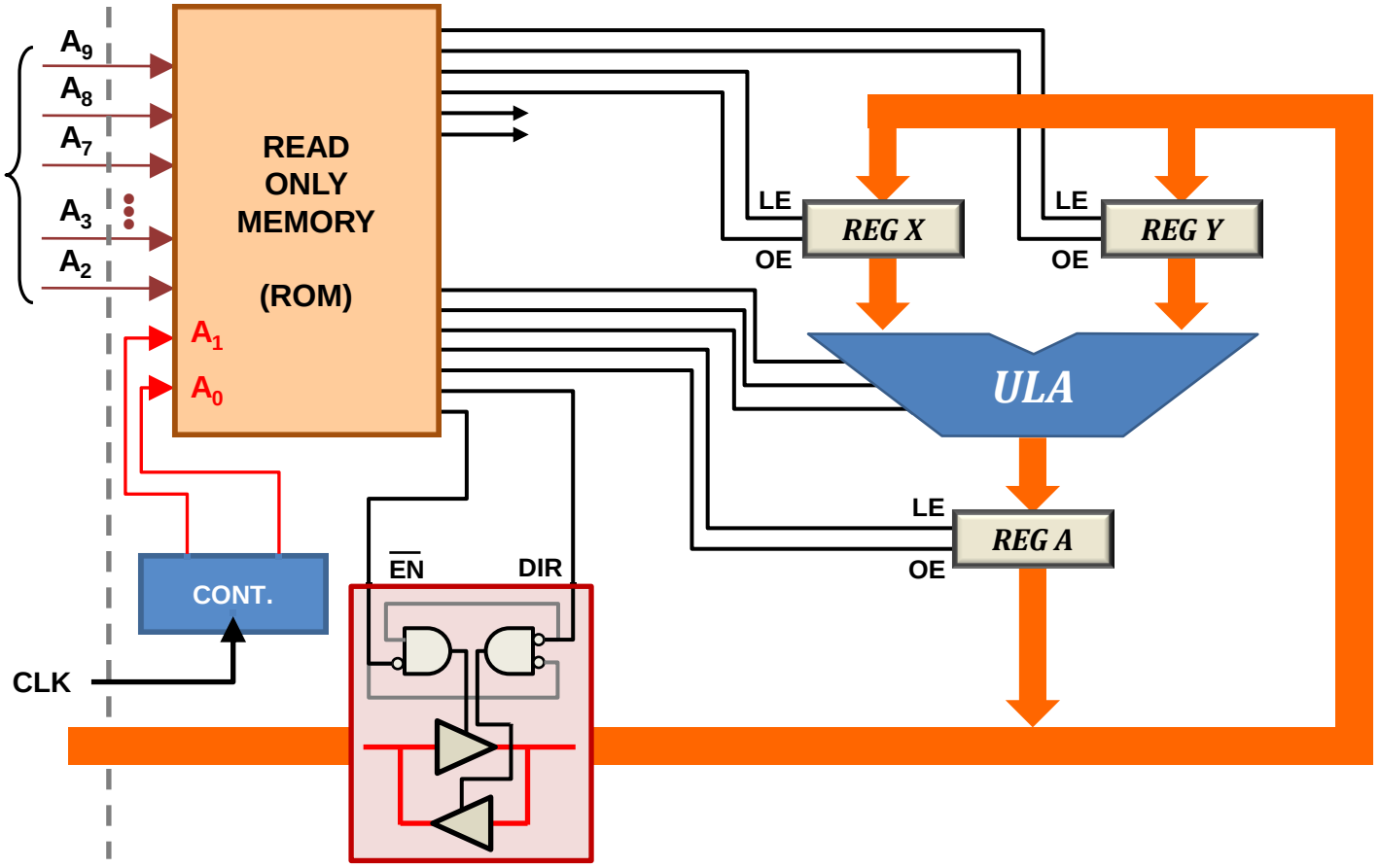
Segundo este raciocínio, quantas ***operações*** de controle (**quantas FSMs**) essa memória contém?

➢ 2^8 bits = 256 FSMs
(ou 256 sequências)



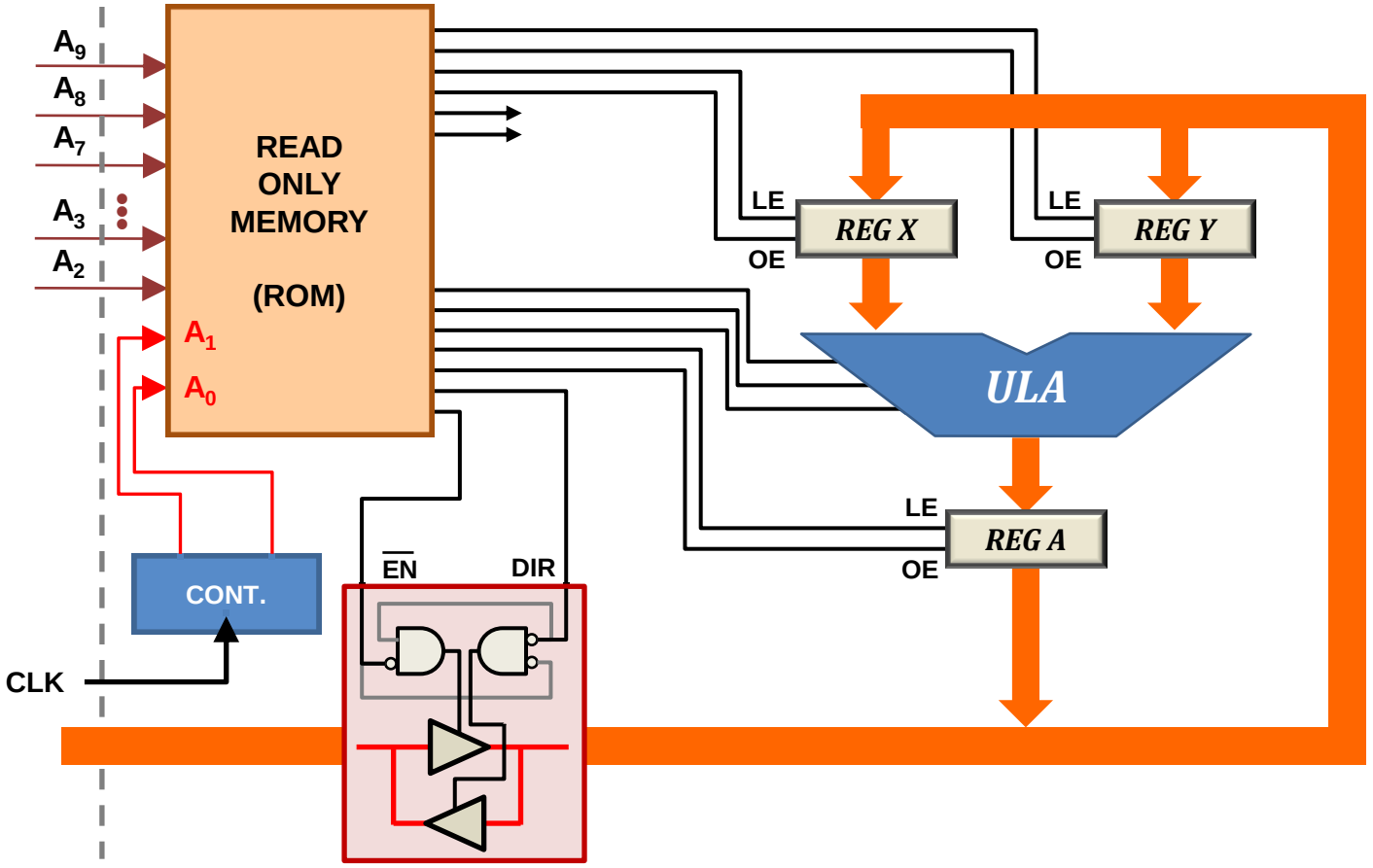
Vamos chamar ***INSTRUÇÃO*** cada FSM que o circuito executa em 4 passos!

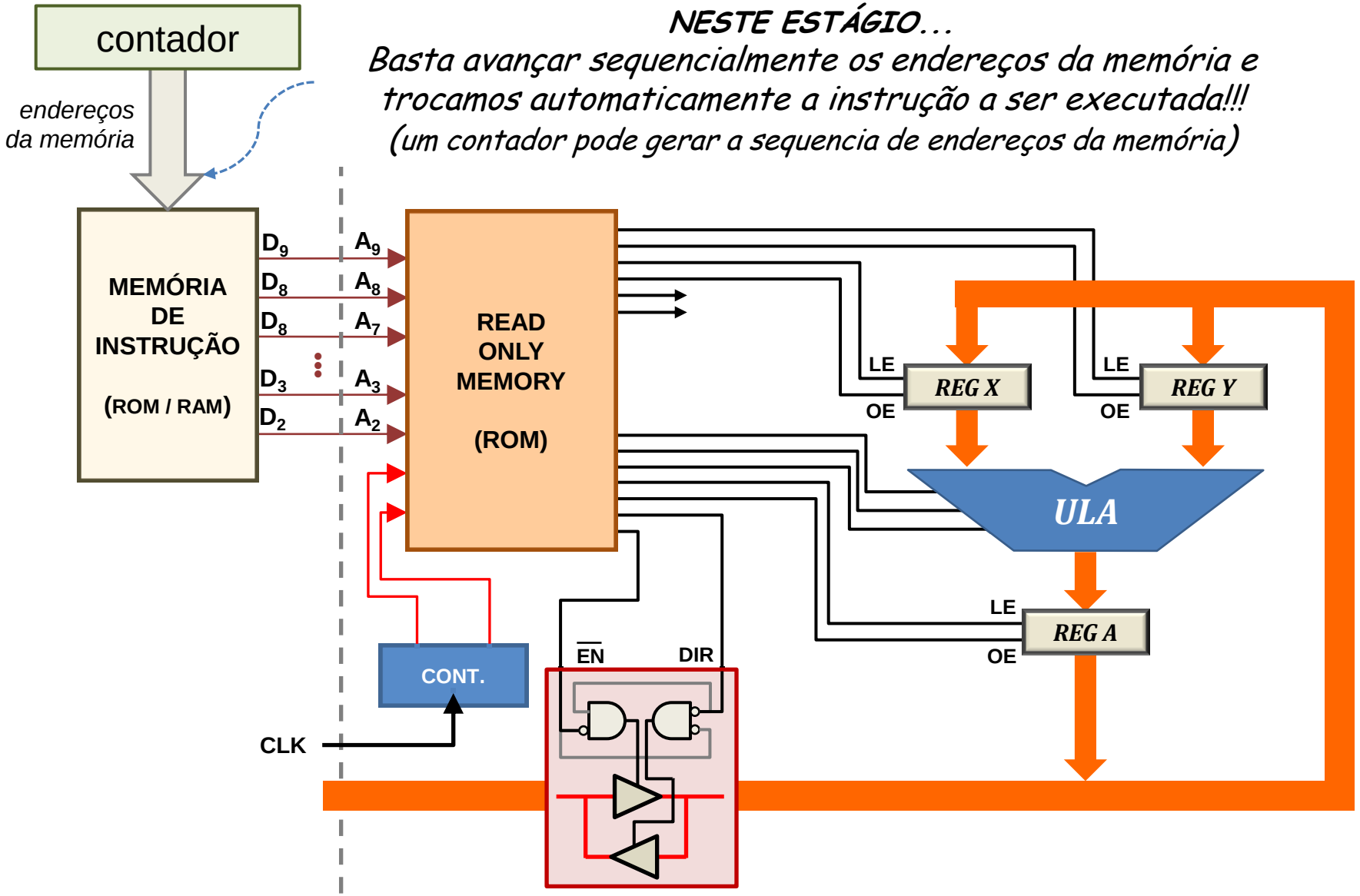
Neste estágio, indicamos externamente qual INSTRUÇÃO queremos, e o circuito executa internamente uma FSM de 4 passos!

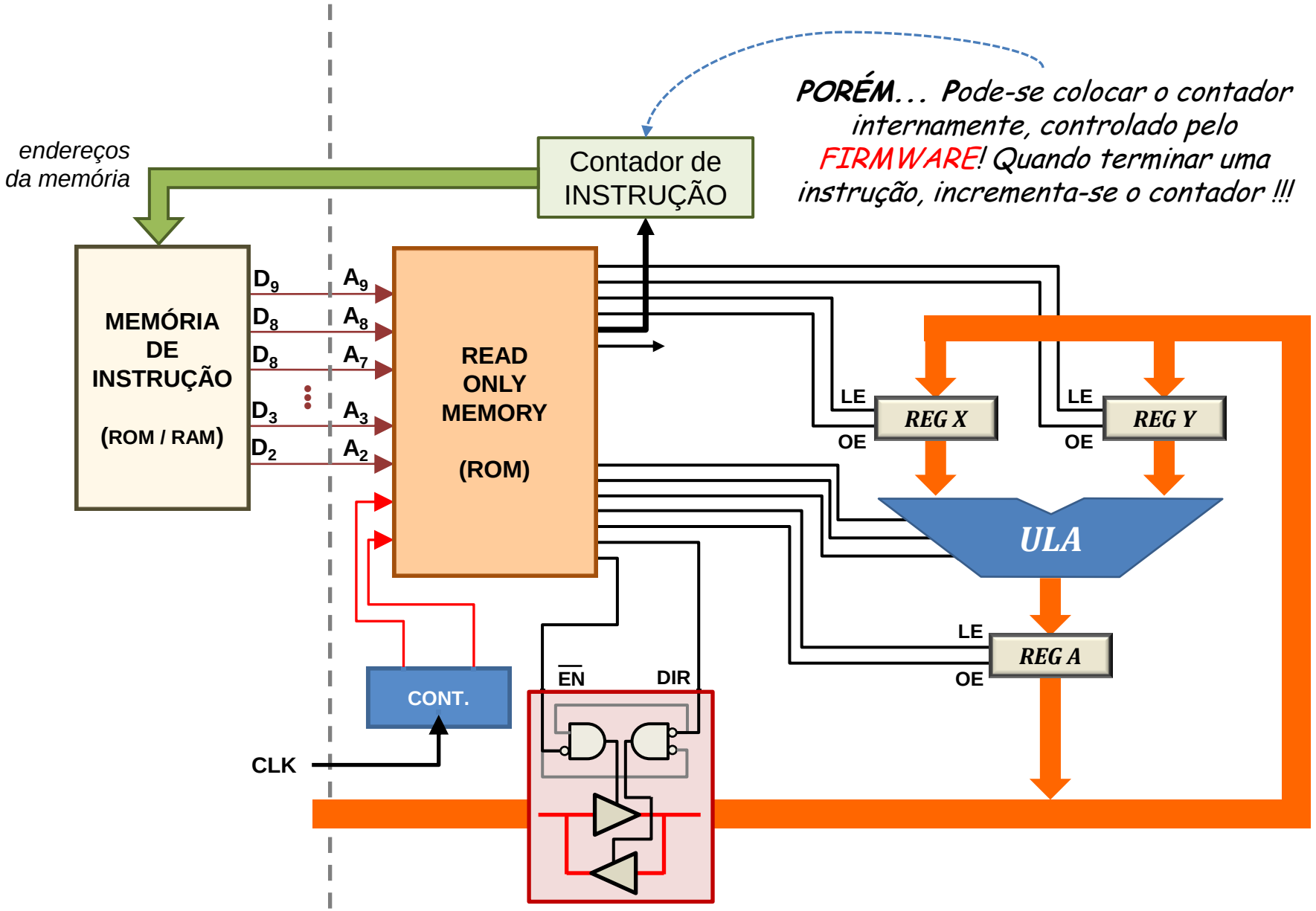


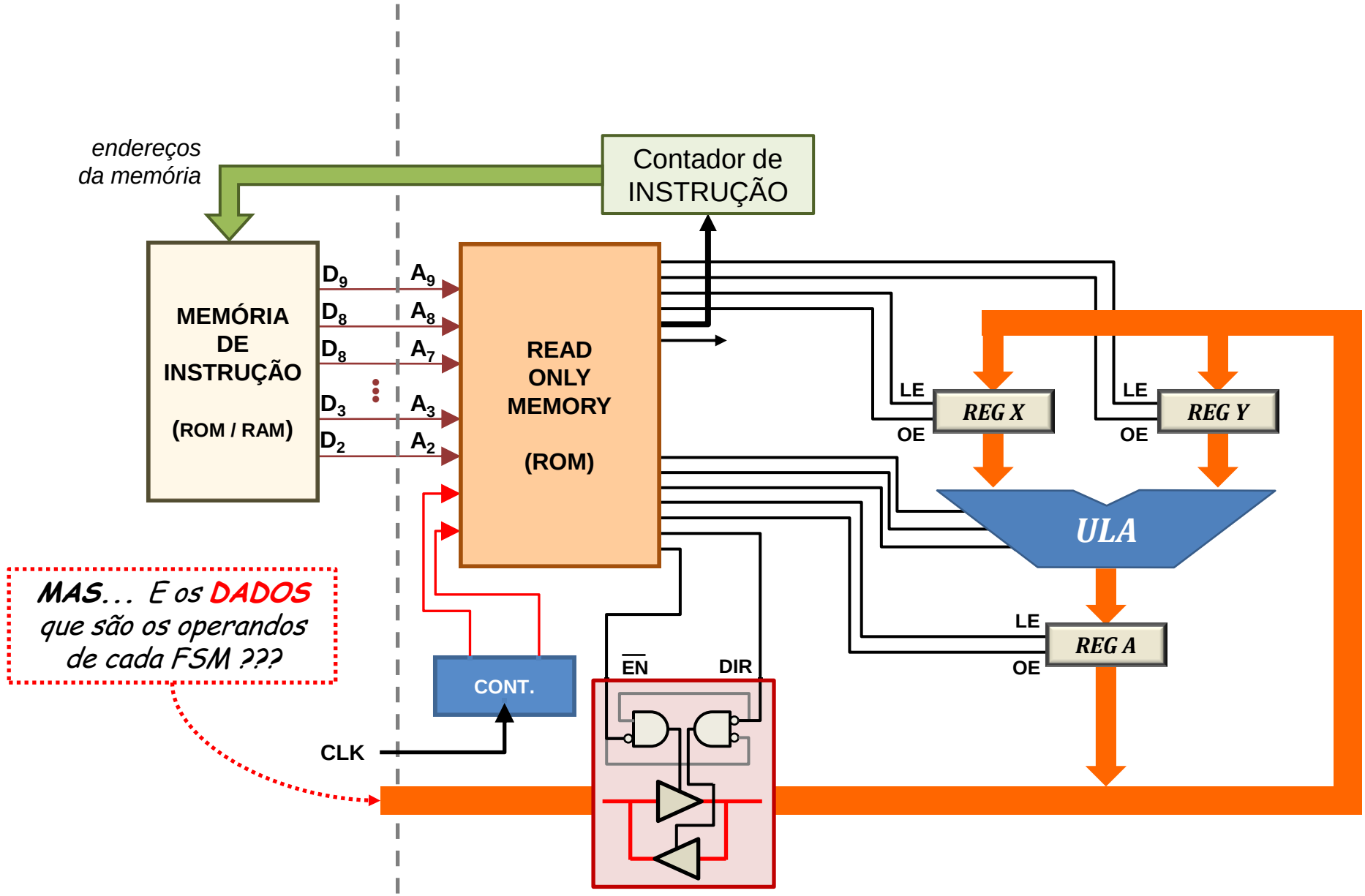
E SE...

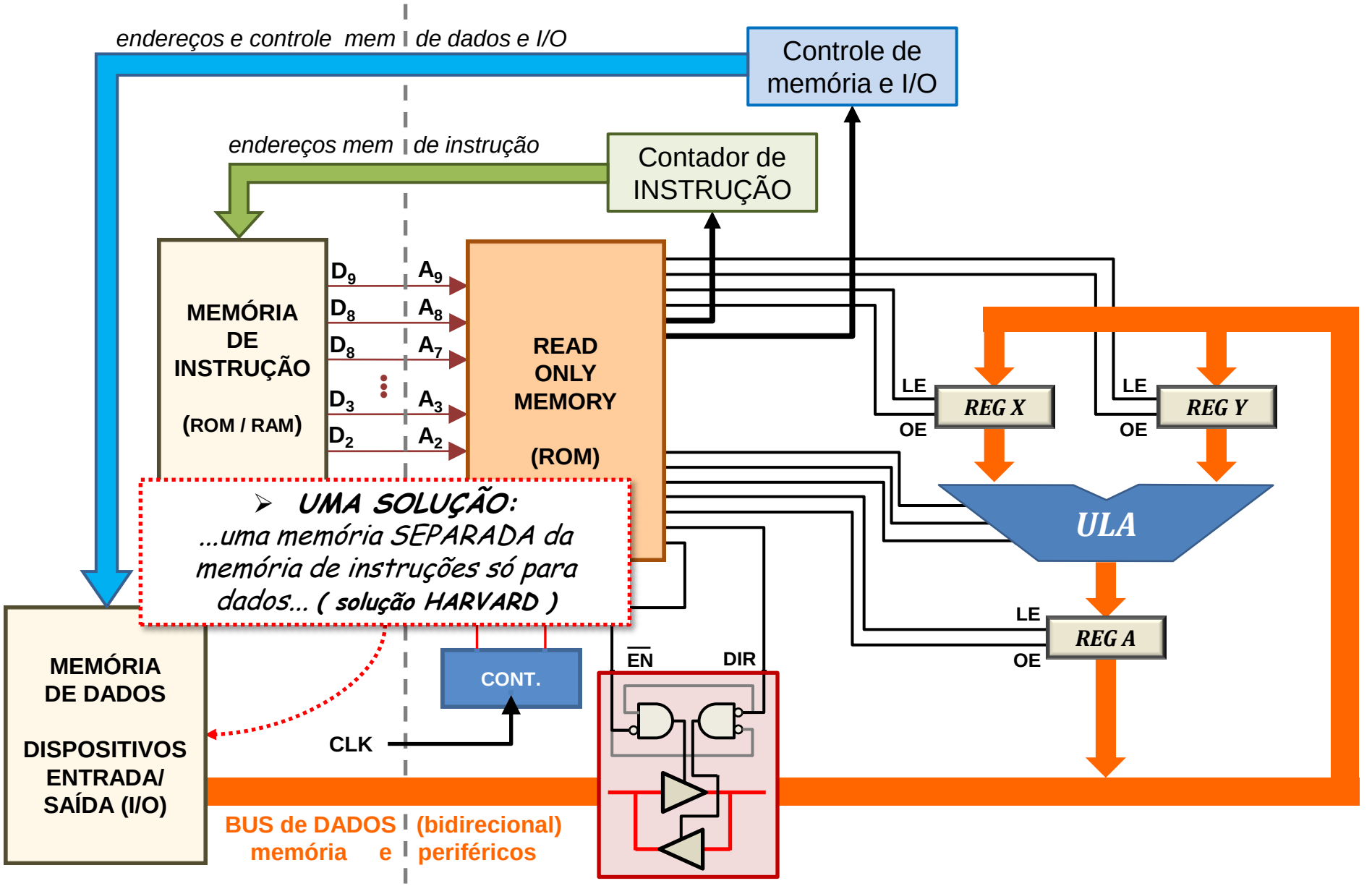
Gravarmos uma série de instruções externamente em uma memória?

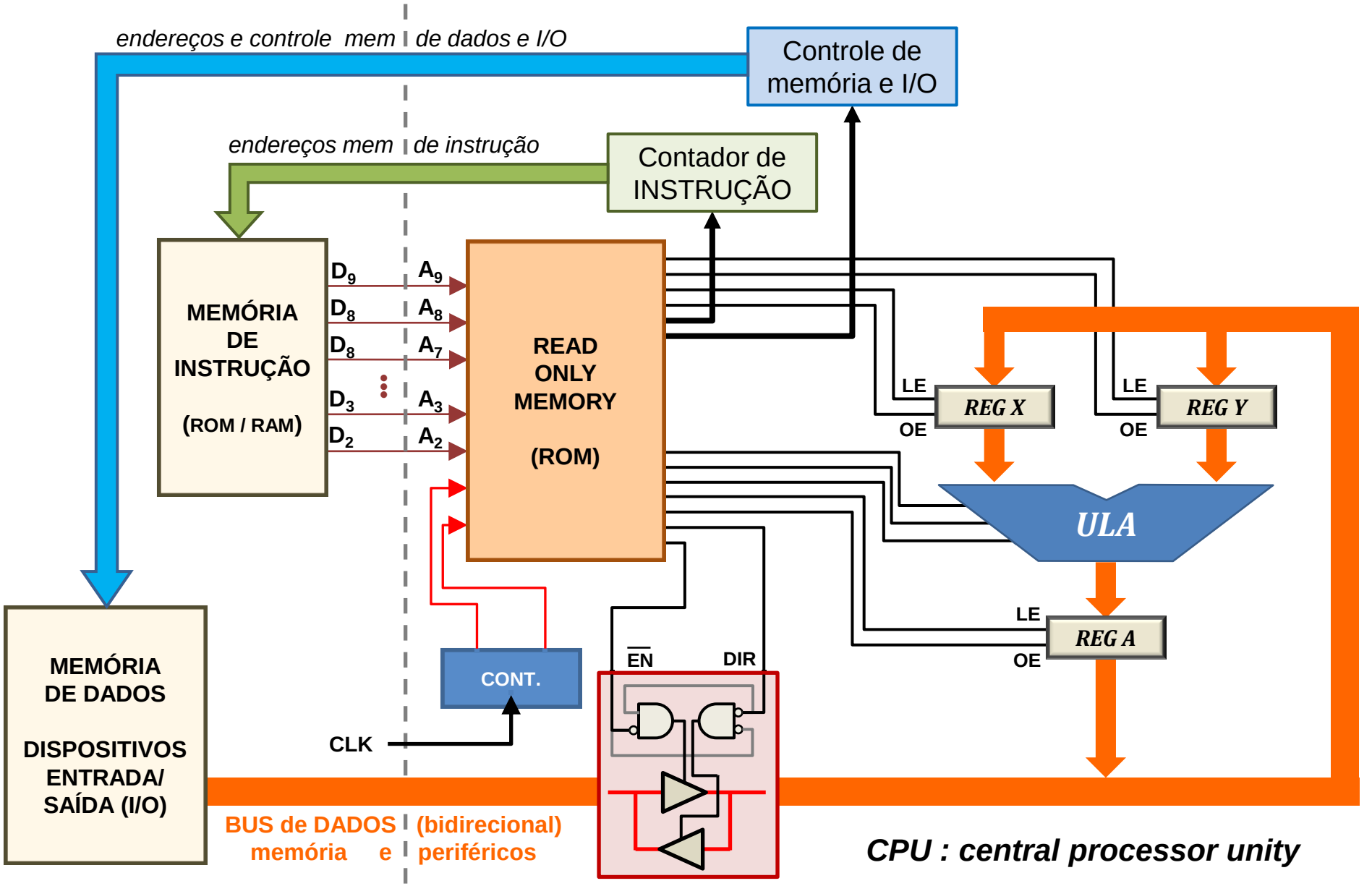



contador
endereços da memória
NESTE ESTÁGIO...
Basta avançar sequencialmente os endereços da memória e trocamos automaticamente a instrução a ser executada!!!
(um contador pode gerar a sequencia de endereços da memória)
MEMÓRIA DE INSTRUÇÃO
(ROM / RAM)
D9
D8
D8
D3
D2
A9
A8
A7
A3
A2
READ ONLY MEMORY
(ROM)
LE
OE
REG X
LE
OE
REG Y
ULA
LE
OE
REG A
CONT.
CLK
EN
DIR

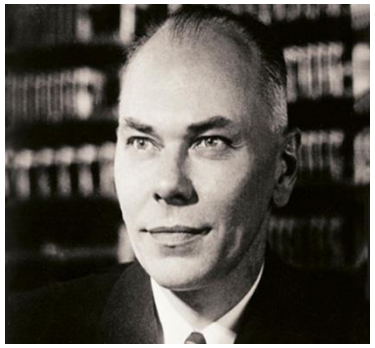

PORÉM... Pode-se colocar o contador internamente, controlado pelo FIRMWARE! Quando terminar uma instrução, incrementa-se o contador !!!
endereços da memória
Contador de INSTRUÇÃO
MEMÓRIA DE INSTRUÇÃO
(ROM / RAM)
D9
D8
D8
D3
D2
A9
A8
A7
A3
A2
READ ONLY MEMORY
(ROM)
LE
OE
REG X
LE
OE
REG Y
ULA
LE
OE
REG A
CONT.
CLK
EN
DIR


endereços da memória
Contador de INSTRUÇÃO
MEMÓRIA DE INSTRUÇÃO
(ROM / RAM)
D9
D8
D8
D3
D2
A9
A8
A7
A3
A2
READ ONLY MEMORY
(ROM)
LE
OE
REG X
LE
OE
REG Y
ULA
LE
OE
REG A
MAS... E os DADOS que são os operandos de cada FSM ???
CONT.
CLK
EN
DIR


endereços e controle mem
de dados e I/O
Controle de memória e I/O
endereços mem
de instrução
Contador de INSTRUÇÃO
MEMÓRIA DE INSTRUÇÃO
(ROM / RAM)
D9
D8
D8
D3
D2
A9
A8
A7
A3
A2
READ ONLY MEMORY
(ROM)
LE
OE
REG X
LE
OE
REG Y
ULA
LE
OE
REG A
➢ UMA SOLUÇÃO:
...uma memória SEPARADA da memória de instruções só para dados... (solução HARVARD)
MEMÓRIA DE DADOS
DISPOSITIVOS ENTRADA/ SAÍDA (I/O)
CONT.
CLK
EN
DIR
BUS de DADOS (bidirecional)
memória e periféricos

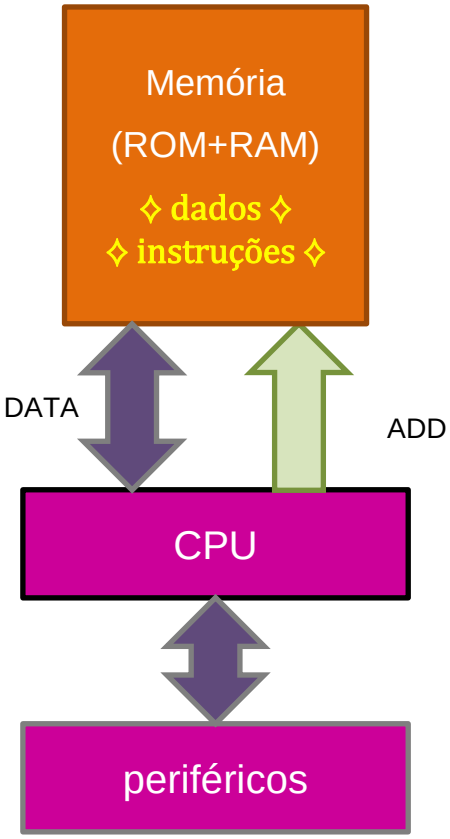

endereços e controle mem
de dados e I/O
Controle de memória e I/O
endereços mem
de instrução
Contador de INSTRUÇÃO
MEMÓRIA DE INSTRUÇÃO
(ROM / RAM)
D9
D8
D8
D3
D2
A9
A8
A7
A3
A2
READ ONLY MEMORY
(ROM)
LE
OE
REG X
LE
OE
REG Y
ULA
LE
OE
REG A
CONT.
CLK
EN
DIR
MEMÓRIA DE DADOS
DISPOSITIVOS ENTRADA/ SAÍDA (I/O)
BUS de DADOS (bidirecional)
memória e periféricos
CPU : central processor unity

John von Neumann



Howard Aiken - Harvard

➢ Arquitetura von Neumann

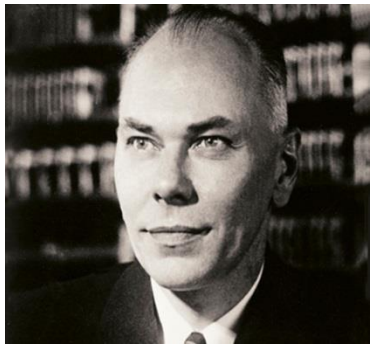


O projeto da Unidade de Controle na CPU deve levar em conta que o ***barramento de dados*** ora transporta ***INSTRUÇÕES*** e ora transporta ***DADOS***, **multiplexados no tempo**.

A memória externa à CPU tem instruções e dados intercalados; portanto, a unidade de controle tem que coordenar quando/onde está buscando instrução e quando/onde está buscando ou gravando dados.

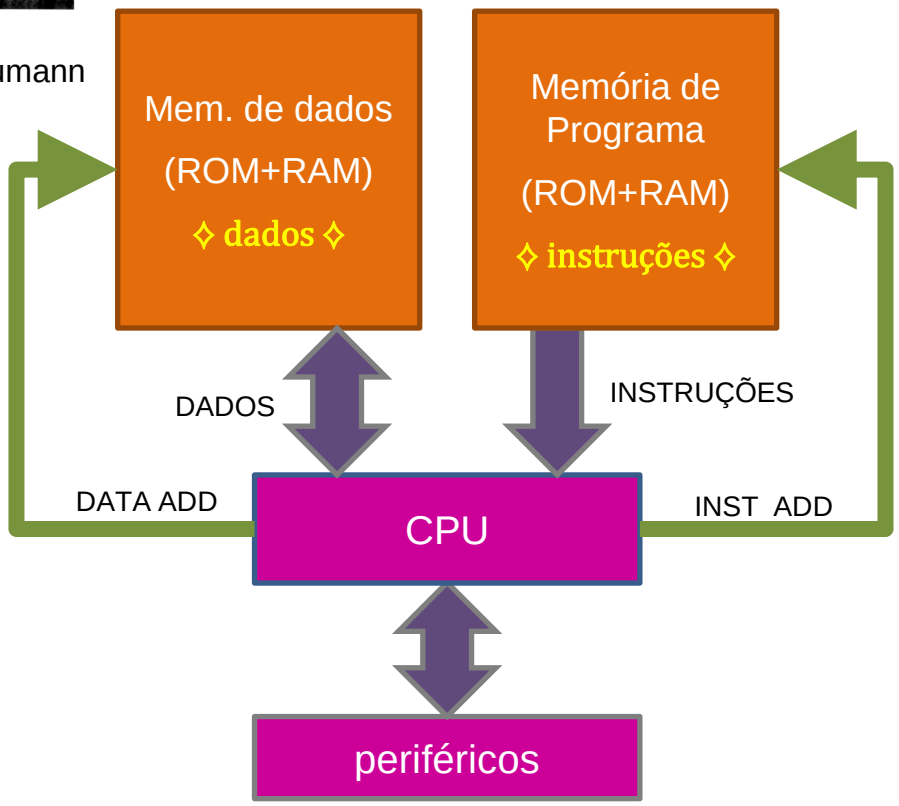
Exemplo: arquiteturas X86, AMD, etc...

John von Neumann



Howard Aiken - Harvard

➢ Arquitetura Harvard



A Unidade de Controle na CPU pode operar os barramentos simultaneamente.

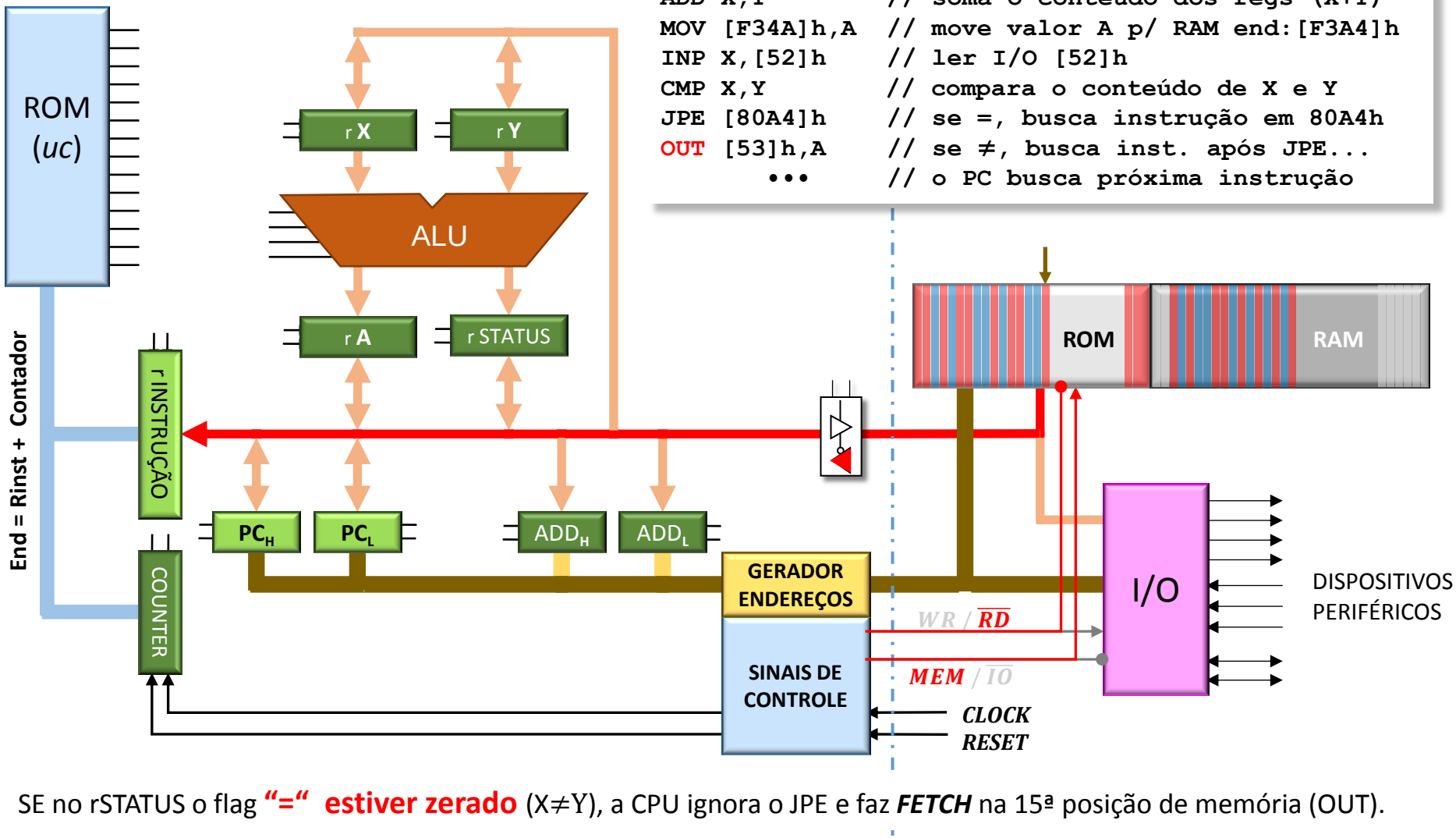
Isso significa que pode ocorrer ***busca de INSTRUÇÕES*** **ao mesmo tempo** em que ocorre ***busca ou gravação*** na memória de **dados**. Operação em "pipeline" pode aumentar a eficiência do processamento.

Por outro lado, implica que a unidade de controle tem que lidar com dois endereçamentos.

Exemplos: ARM, PIC, AVR (Arduino), etc...

μProcessador (CPU)

```
CLF              // (Clear Flags) zera reg. de flags
MOV X,80h        // move 80 Hexa p/ o registrador X
MOV Y,2Ah        // move 2A Hexa p/ registrador Y
ADD X,Y          // soma o conteúdo dos regs (X+Y)
MOV [F34A]h,A    // move valor A p/ RAM end:[F3A4]h
INP X,[52]h      // ler I/O [52]h
CMP X,Y          // compara o conteúdo de X e Y
JPE [80A4]h      // se =, busca instrução em 80A4h
OUT [53]h,A      // se ≠, busca inst. após JPE...
     •••         // o PC busca próxima instrução
```



SE no rSTATUS o flag **"="** **estiver zerado** (X≠Y), a CPU ignora o JPE e faz ***FETCH*** na 15ª posição de memória (OUT).

Conceitos e memorial da aula de hoje:

- Revisão sobre memória, ULA, registradores, transceptores, ULA;
- Conceito de Unidade de Controle – ***FIRMWARE*** ;
- Unidade Central de Processamento;
- Duas arquiteturas predominantes em processadores:
- ✓ *Arquitetura* ***Harvard*** (duas memórias – uma para dados e outra para instruções)
- ✓ Arquitetura ***von Neumann*** (memória única – compartilha dados e instruções)
- Um exemplo de processador von Neuman;
- Funcionamento do processador von Neumann (pequeno programa);
- Código de Máquina, mnemônicos, linguagem Assembly.