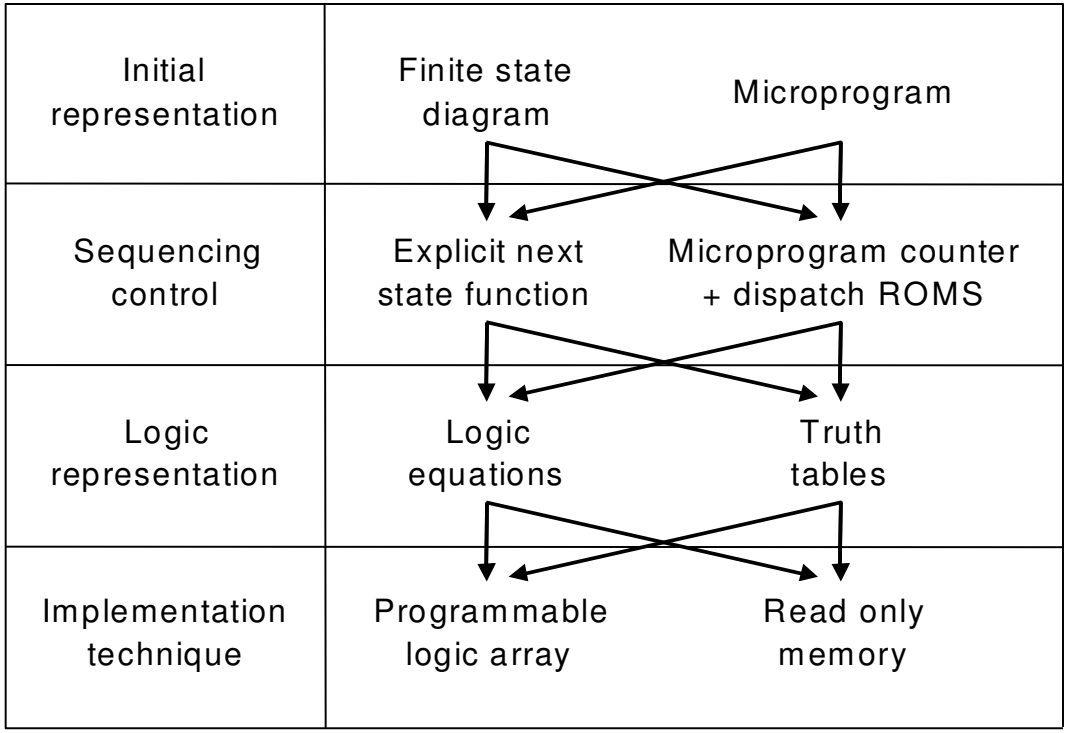
O Processador: Via de Dados e Controle (Parte C: microprogramação)

Possibilidades para o projeto de UCs



- Ler “Historical perspective and further reading”
- RISC x CISC
- Controle
  - hardwired
  - microprogramado (firmware)

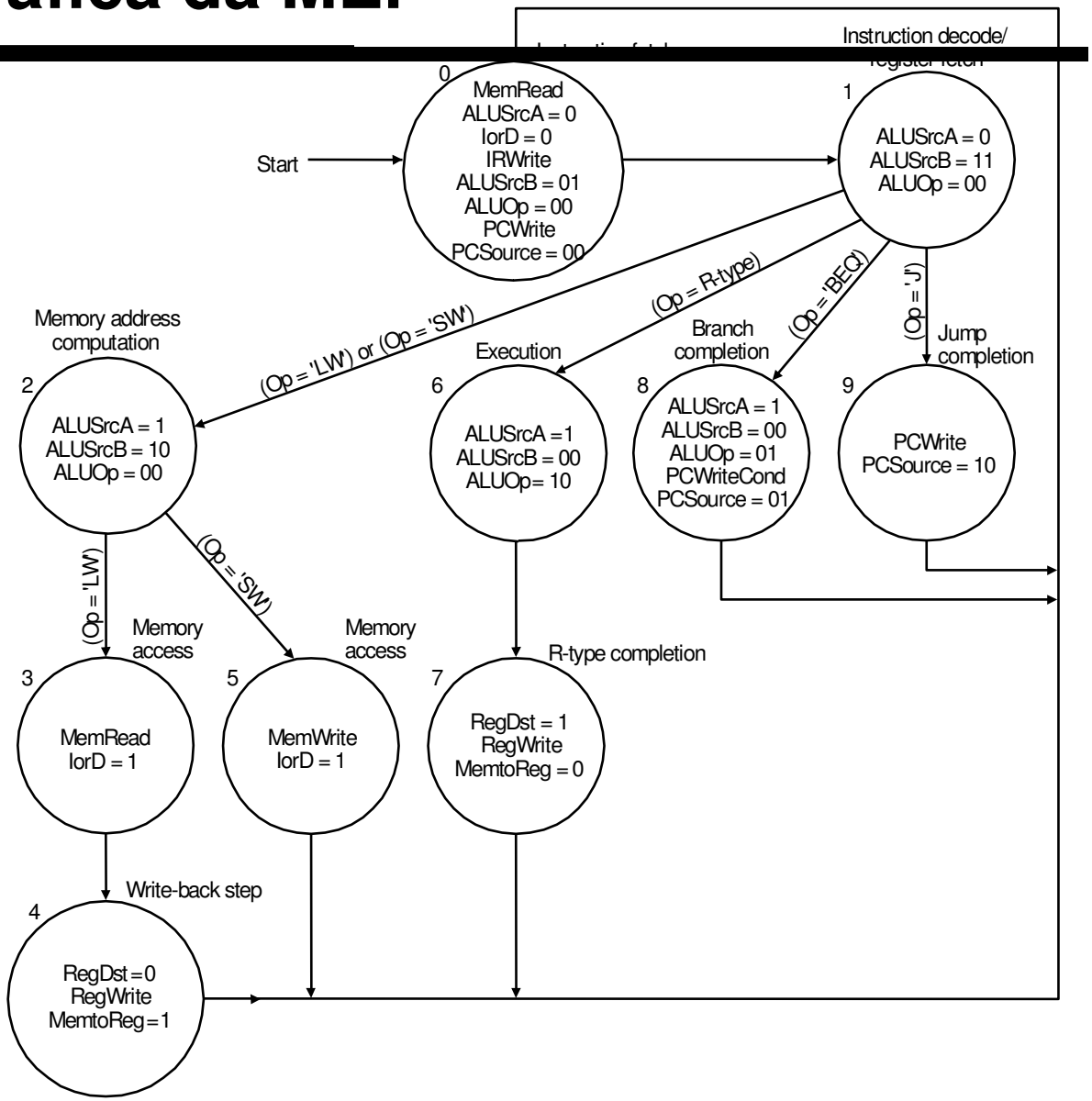
Implementando o controle

- **Valores dos sinais de controle dependem de:**
  - **Qual instrução está sendo executada**
  - **Que passo está sendo processado**

- **Especificar a máquina de estados finitos graficamente, ou**
- **Usar microprogramação**

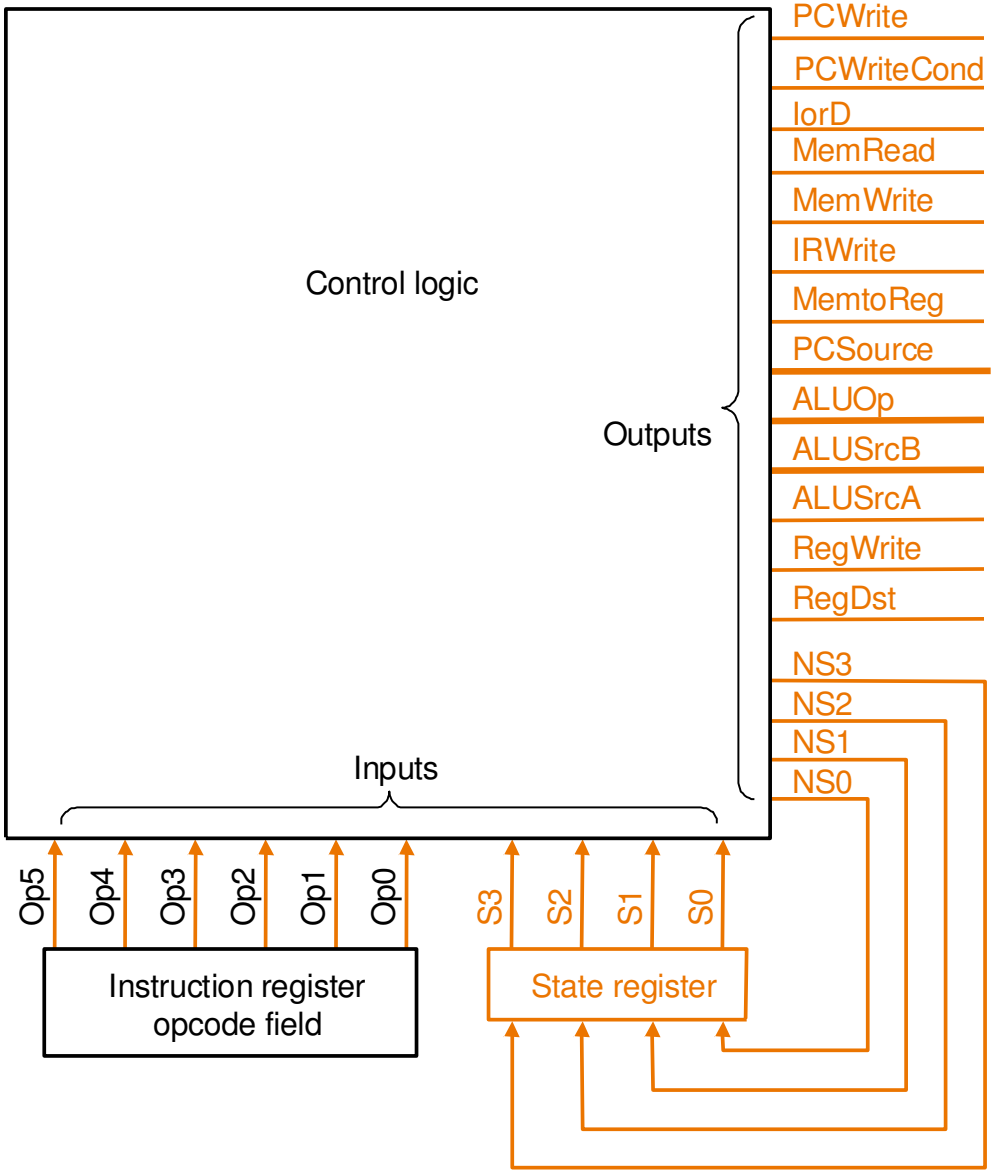
- **Implementação pode ser derivada da especificação**

Especificação gráfica da MEF

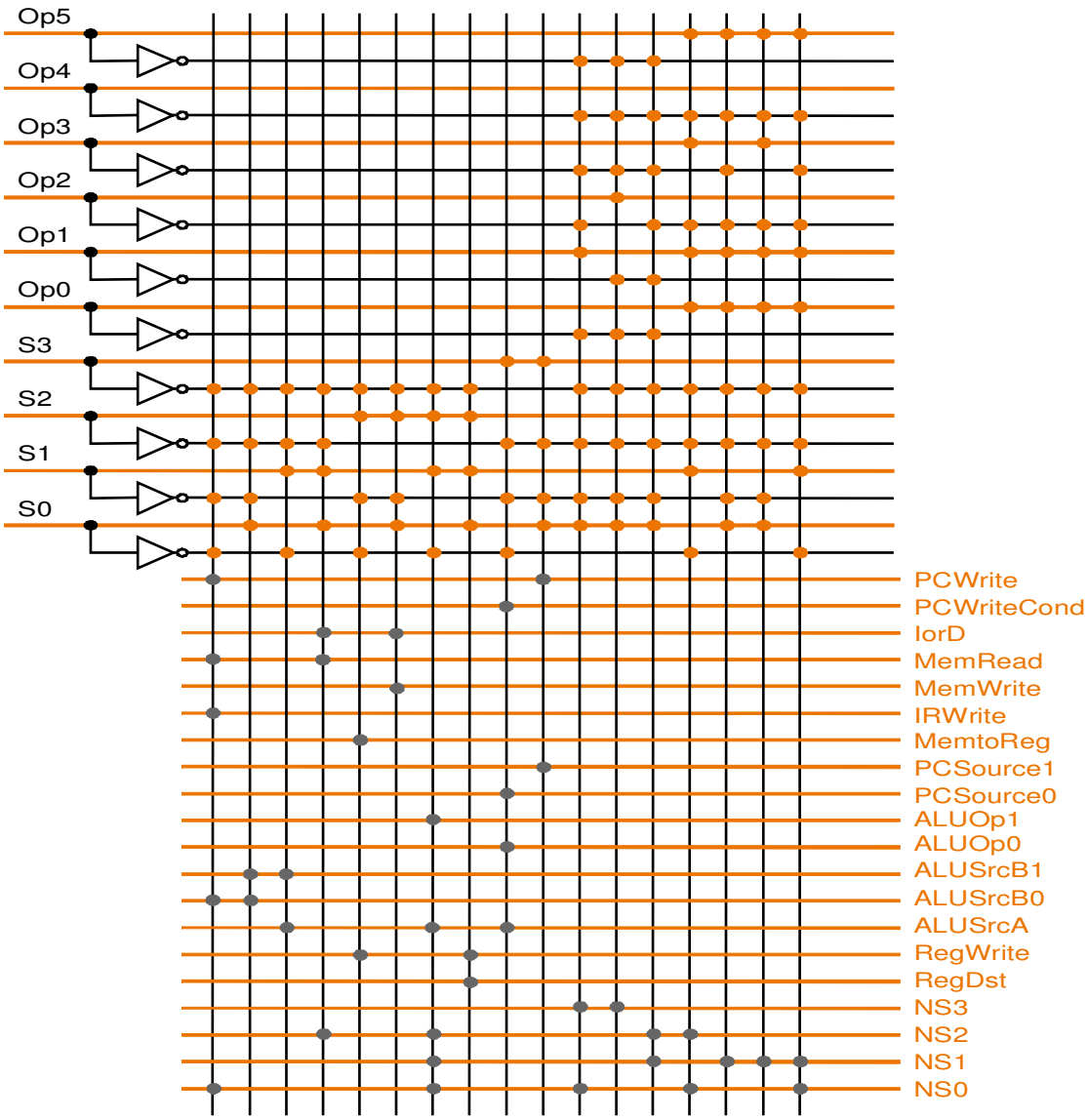


- **Quantos bits de estado serão necessários?**

MEF para controle



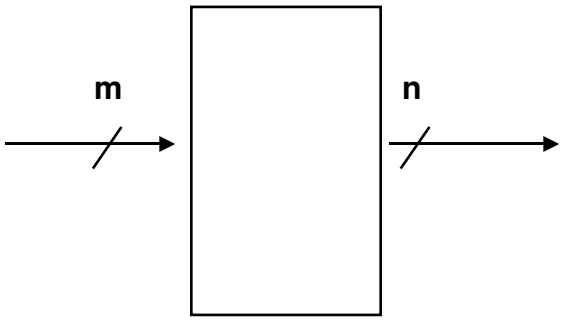
Implementação via PLA



Implementação via ROM

- **ROM = "Read Only Memory"**

- **Uma ROM pode ser usada para implementar uma tabela-verdade**
  - **Se o endereço possui m-bits, pode-se endereçar 2^m entradas na ROM.**
  - **A saída são os bits de dados que o endereço aponta**



0	0	0	0	0	1	1
0	0	1	1	1	0	0
0	1	0	1	1	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	1
1	1	0	0	1	1	0
1	1	1	0	1	1	1

m é a "altura", e n é a "largura"

Implementação via ROM

- **Quantas entradas?**
  6 bits para opcode, 4 bits para estado = 10 bits de endereço
  (i.e., 2^{10} = 1024 endereços diferentes)
- **Quantas saídas?**
  16 saídas para controle da via de dados, 4 bits de estado= 20 bits de saída

- **ROM é 2^{10} x 20 = 1K x 20 bits**

- **Ocupa muita área!**

ROM vs PLA

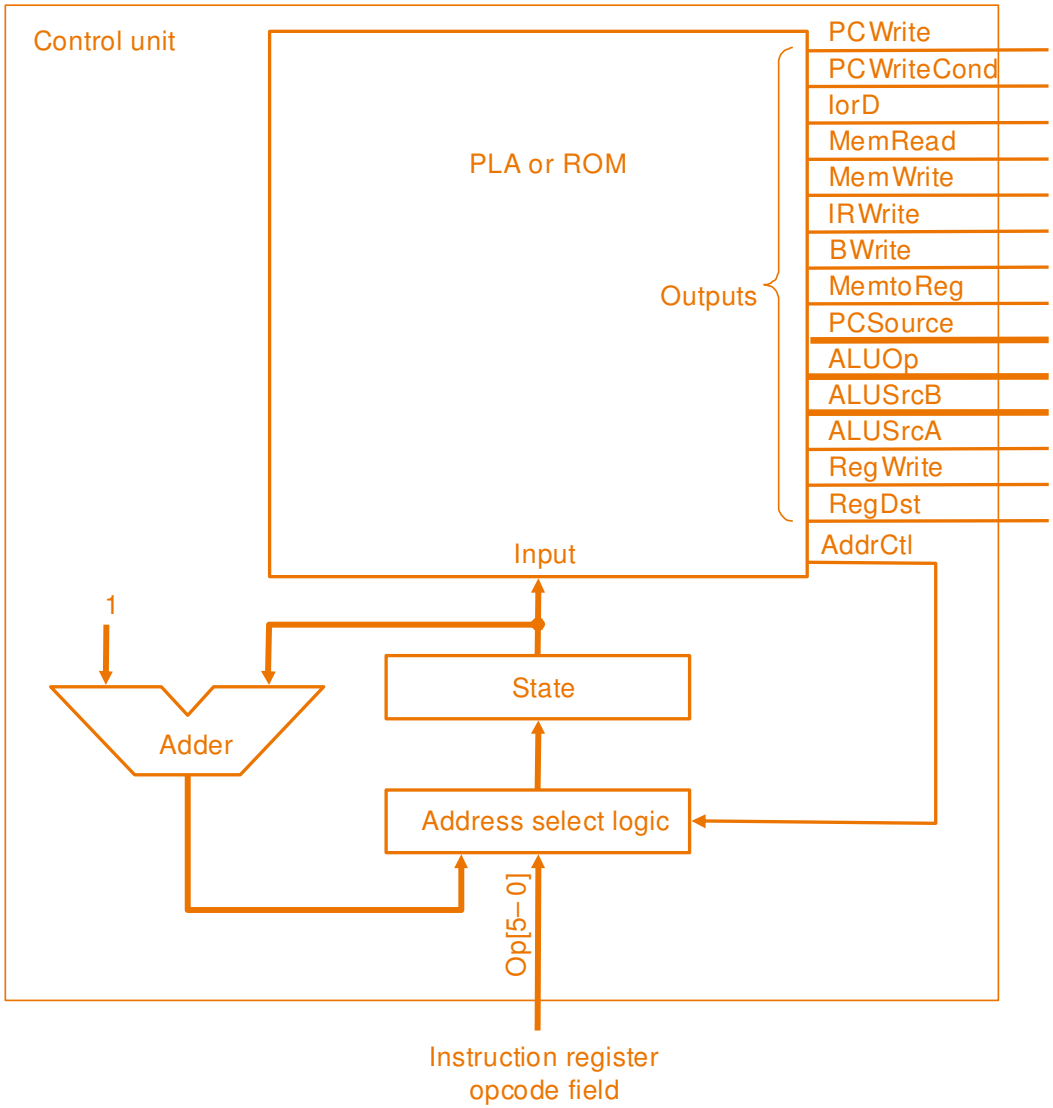
- **Divisão da tabela em duas partes**
  - **— 4 bits de estado indicam 16 saídas, 2^4 x 16 bits da ROM**
  - **— 10 bits tell you the 4 next state bits, 2^{10} x 4 bits da ROM**
  - **— Total: 4.3K bits de ROM**

- **PLA é muito menor**
  - **— Pode compartilhar termos de produtos**
  - **— Necessita apenas de entradas que produzem uma saída ativa**
  - **— Pode levar em conta sinais do tipo don't care**

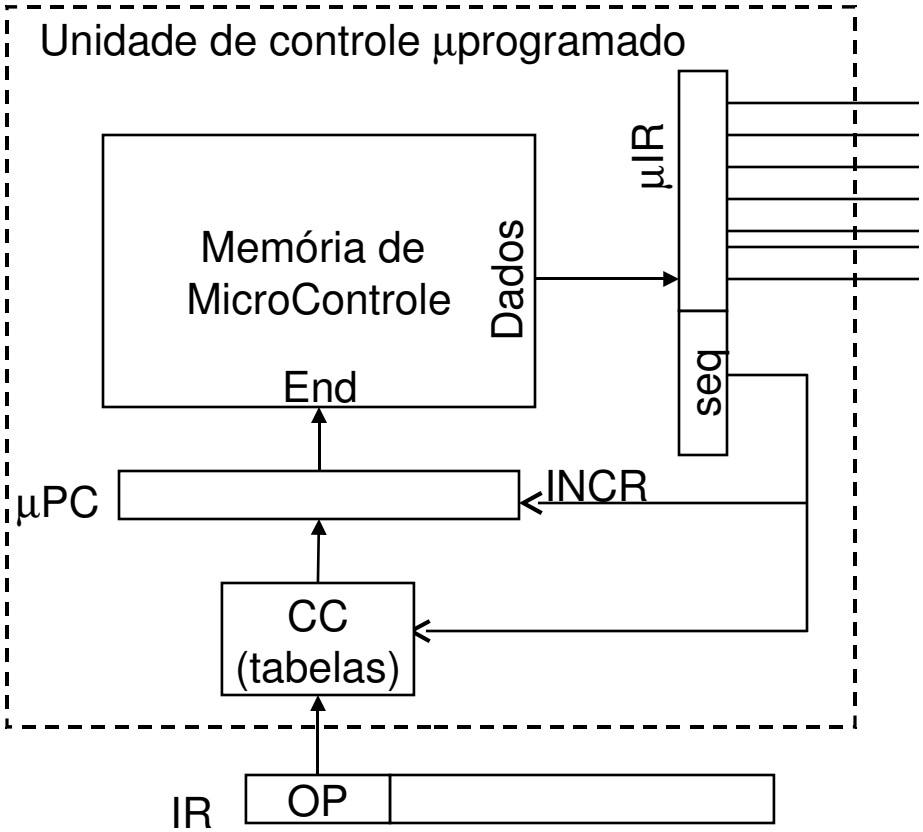
- **Tamanho é (#inputs × #product-terms) + (#outputs × #product-terms)**
  Para o nosso exemplo = (10x17)+(20x17) = 460 células de PLA

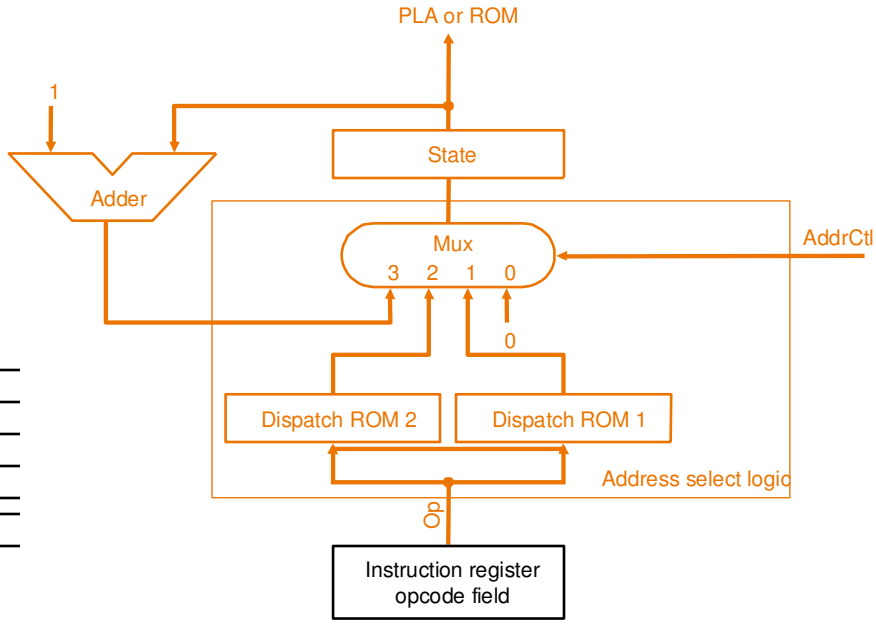
Outro estilo de implementação

- **Instruções complexas: o "próximo estado" é o estado atual + 1**



Visão geral

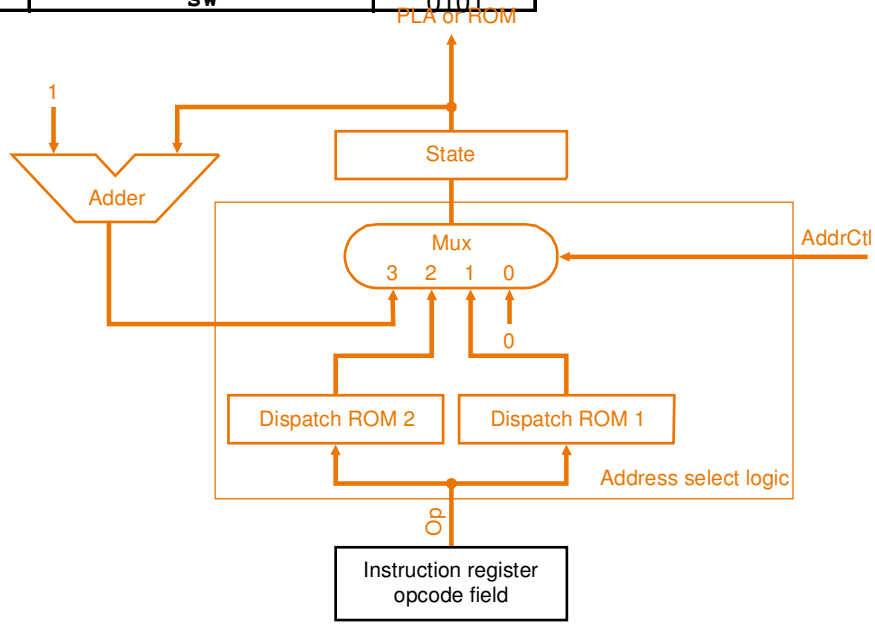




controle

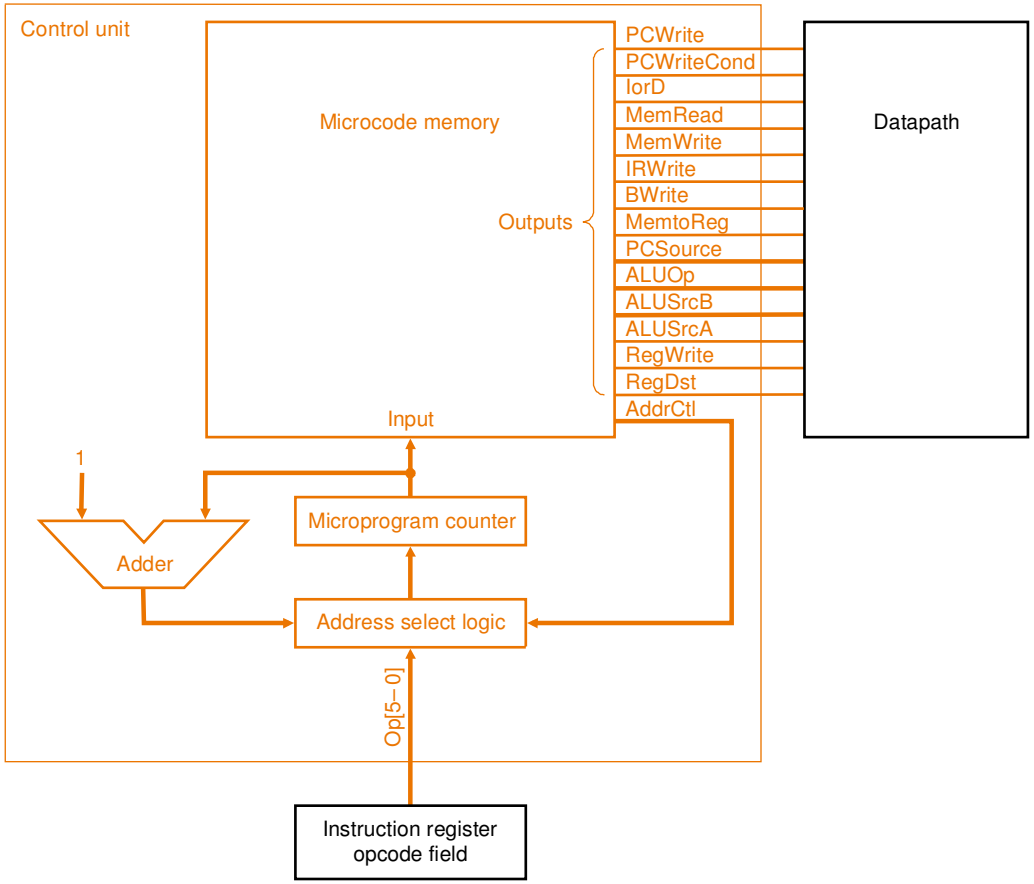
Dispatch ROM 1		
Op	Opcode name	Value
000000	R-format	0110
000010	jmp	1001
000100	beq	1000
100011	lw	0010
101011	sw	0010

Dispatch ROM 2		
Op	Opcode name	Value
100011	lw	0011
101011	sw	0101



State number	Address-control action	Value of AddrCtl
0	Use incremented state	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Use incremented state	3
4	Replace state number by 0	0
5	Replace state number by 0	0
6	Use incremented state	3
7	Replace state number by 0	0
8	Replace state number by 0	0
9	Replace state number by 0	0

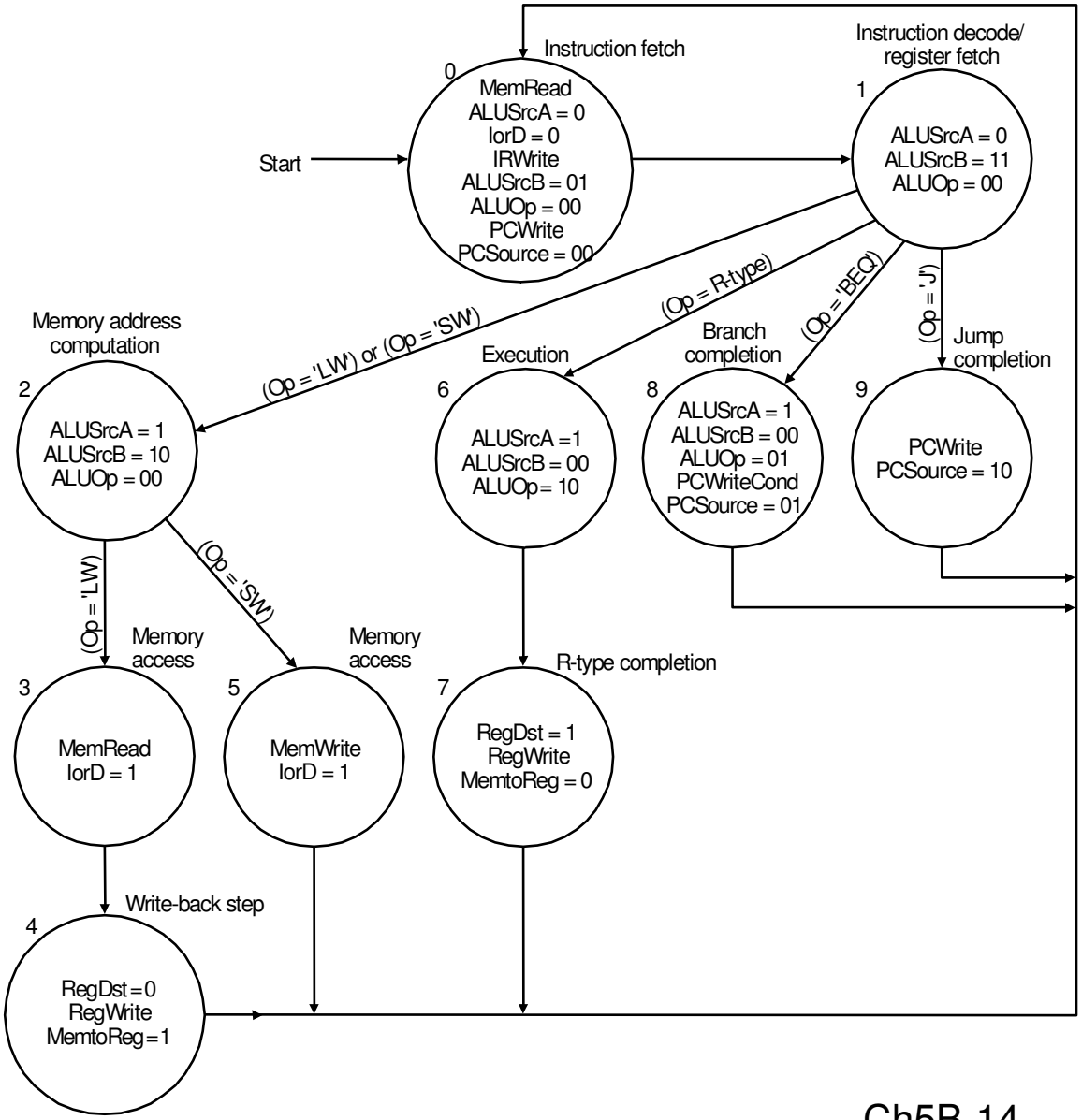
Microprogramação



- **O que são as microinstruções?**

Diagrama de transição de estados

Estado	Address-control action	seq
0	Use incremented state	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Use incremented state	3
4	Replace state number by 0	0
5	Replace state number by 0	0
6	Use incremented state	3
7	Replace state number by 0	0
8	Replace state number by 0	0
9	Replace state number by 0	0



Um microprograma horizontal

Estado	Address-control action	seq
0	Use incremented state	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Use incremented state	3
4	Replace state number by 0	0
5	Replace state number by 0	0
6	Use incremented state	3
7	Replace state number by 0	0
8	Replace state number by 0	0
9	Replace state number by 0	0

	IorD	IRWR	MemRD	MemWR	PCWR	PCWR-cond	Mem2Reg	RegDst	RegWR	AluSrcA	AluSrcB	AluOP	PCSrc	Desvio	Obs
Fetch	0	1	1		1					0	01	00	00	Seq	Lê instrucão; PC <- PC+4
										0	11	00		Dispatch1	ALUout<- end de desvio
LWSW1										1	10	00		Dispatch2	end efetivo
LW2	1		1											Seq	Lê memória
							1	0	1					Fetch	Write Back
SW2	1			1										Fetch	Escreve na Mem.
RFormat1										1	00	10		Seq	Executa ALU
							0	1	1					Fetch	Write Back
Beq1						1				1	00	01	01	Fetch	Desvio condicional
Jump1					1								10	Fetch	Desvio incondicional

Microprogramação

- **Uma metodologia de especificação**
  - **Apropriada se centenas de opcodes, modos, ciclos, etc.**
  - **Sinais especificados simbolicamente usando microinstruções**

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read (nada)			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

- ***Duas implementações da mesma arquitetura possuem o mesmo microcódigo?***

Formato das microinstruções

Field name	Value	Signals active	Comment
ALU control	Add	ALUOp = 00	Cause the ALU to add.
	Subt	ALUOp = 01	Cause the ALU to subtract; this implements the compare for branches.
	Func code	ALUOp = 10	Use the instruction's function code to determine ALU control.
SRC1	PC	ALUSrcA = 0	Use the PC as the first ALU input.
	A	ALUSrcA = 1	Register A is the first ALU input.
SRC2	B	ALUSrcB = 00	Register B is the second ALU input.
	4	ALUSrcB = 01	Use 4 as the second ALU input.
	Extend	ALUSrcB = 10	Use output of the sign extension unit as the second ALU input.
	Extshft	ALUSrcB = 11	Use the output of the shift-by-two unit as the second ALU input.
Register control	Read		Read two registers using the rs and rt fields of the IR as the register numbers and putting the data into registers A and B.
	Write ALU	RegWrite, RegDst = 1, MemtoReg = 0	Write a register using the rd field of the IR as the register number and the contents of the ALUOut as the data.
	Write MDR	RegWrite, RegDst = 0, MemtoReg = 1	Write a register using the rt field of the IR as the register number and the contents of the MDR as the data.
Memory	Read PC	MemRead, IorD = 0	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	MemRead, IorD = 1	Read memory using the ALUOut as address; write result into MDR.
	Write ALU	MemWrite, IorD = 1	Write memory using the ALUOut as address, contents of B as the data.
PC write control	ALU	PCSource = 00 PCWrite	Write the output of the ALU into the PC.
	ALUOut-cond	PCSource = 01, PCWriteCond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	jump address	PCSource = 10, PCWrite	Write the PC with the jump address from the instruction.
Sequencing	Seq	AddrCtl = 11	Choose the next microinstruction sequentially.
	Fetch	AddrCtl = 00	Go to the first microinstruction to begin a new instruction.
	Dispatch 1	AddrCtl = 01	Dispatch using the ROM 1.
	Dispatch 2	AddrCtl = 10	Dispatch using the ROM 2.

Codificação x Não codificação

- **Não codificado**
  - **1 bit para cada operação da via de dados**
  - **Mais rápido, requer mais memória (lógica)**
  - **Usado pelo Vax 780 — incríveis 400K de memória!**
- **Codificado**
  - **Enviar microinstruções e obter sinais de controle**
  - **Usa menos memória mas é mais lento**
- **Contexto histórico de CISC:**
  - **Muita lógica de controle para colocar em um único chip**
  - **Utilizar uma ROM (ou mesmo uma RAM) para manter o microcódigo**
  - **A adição de novas instruções é mais simples**

Microcódigo

- A distinção entre especificação e implementação é, algumas vezes, difícil de encontrar
- Vantagens da especificação:
  - Fácil de projetar e escrever
  - Projetar arquitetura e microcódigo em paralelo
- Vantagens da implementação (ROM)
  - Fácil de modificar uma vez que os valores estão em uma memória
  - Pode emular outras arquiteturas
  - Pode fazer uso de registradores internos
- Desvantagens da implementação :
  - Controle é implementado no mesmo chip como o processador
  - ROM não é mais rápida que uma RAM