



Universidade de Brasília

Departamento de Ciência da Computação

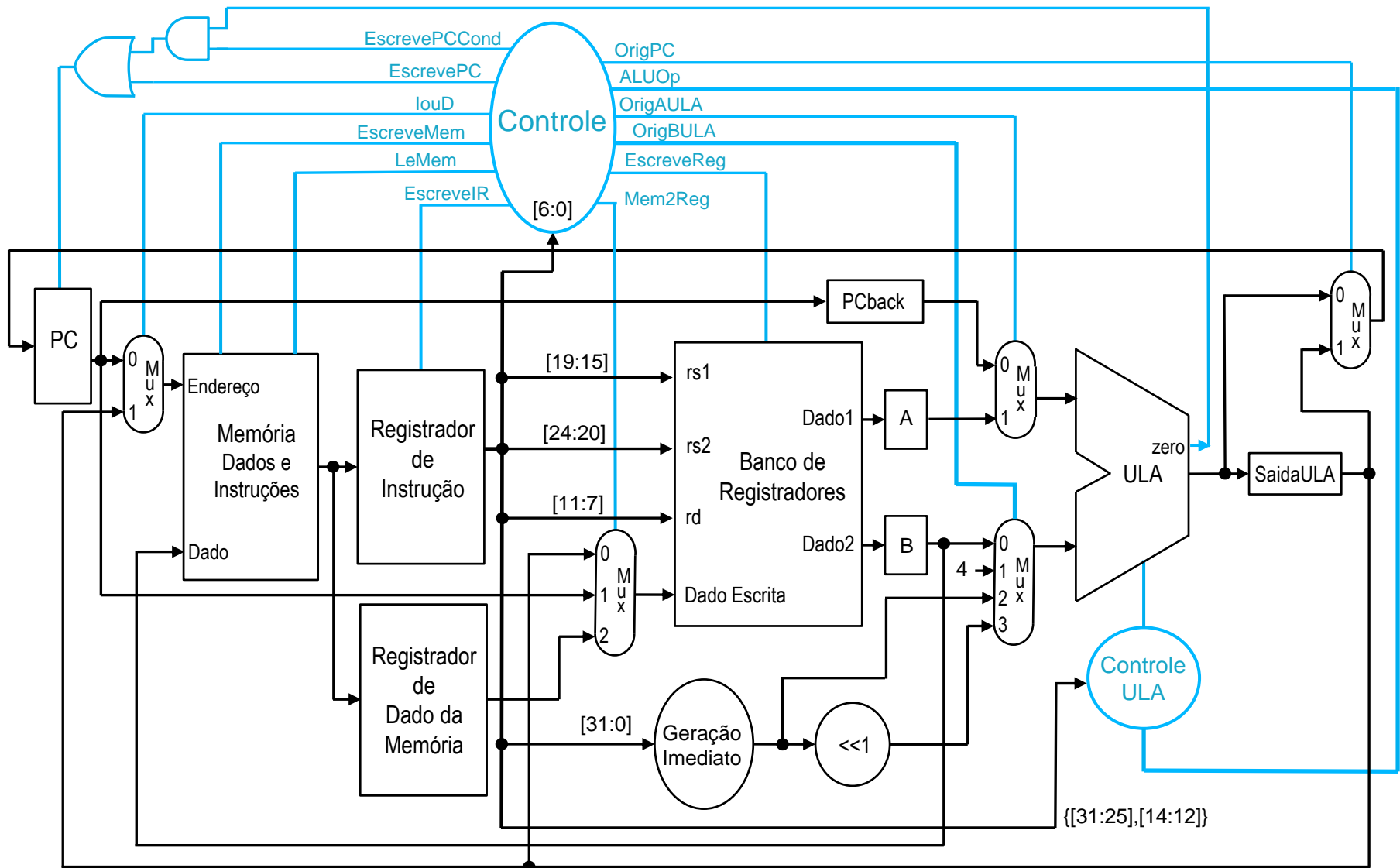
Aula 15

Implementação RISC-V

Multiciclo – Unidade de Controle



Caminho de Dados Multiciclo





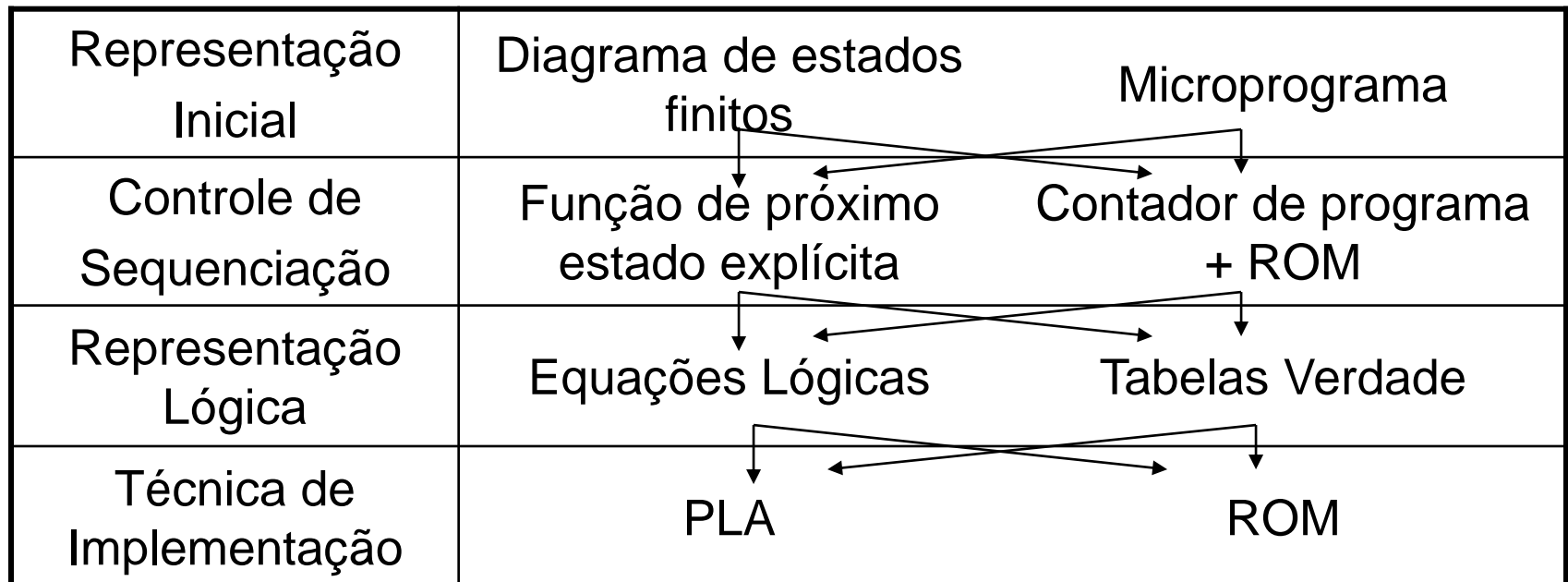
Resumo Controle Multiciclo

Etapa	Tipo-R	Acesso à Memória	Desvios Condicionais	Desvios Incondicionais
Busca da Instrução	$IR \leftarrow Mem[PC]$ $PC_{back} \leftarrow PC$ $PC \leftarrow PC + 4$			
Decodificação, Leitura dos registradores	$A \leftarrow Reg[IR[19:15]]$ $B \leftarrow Reg[IR[24:20]]$ $SaidaULA \leftarrow PC + imm \ll 1$			
Execução, cálculo do endereço	$SaidaULA \leftarrow A \text{ op } B$	$SaidaULA \leftarrow A + imm$	Se $(A == B)$ $PC \leftarrow SaidaULA$	$Reg[IR[11:7]] \leftarrow PC + 4$ $PC \leftarrow SaidaULA$
Acesso à memória, conclusão tipo-R	$Reg[IR[11:7]] \leftarrow SaidaULA$	Load: $MDR \leftarrow Mem[SaidaULA]$ Store: $Mem[SaidaULA] \leftarrow B$		
Conclusão lw		Load: $Reg[IR[11:7]] \leftarrow MDR$		



Projeto do Controle Multiciclo

- Controle feito em uma série de etapas
- Técnicas de Implementação:
 - Máquinas de Estado Finito
 - Microprogramação (usado nos processadores CISC)

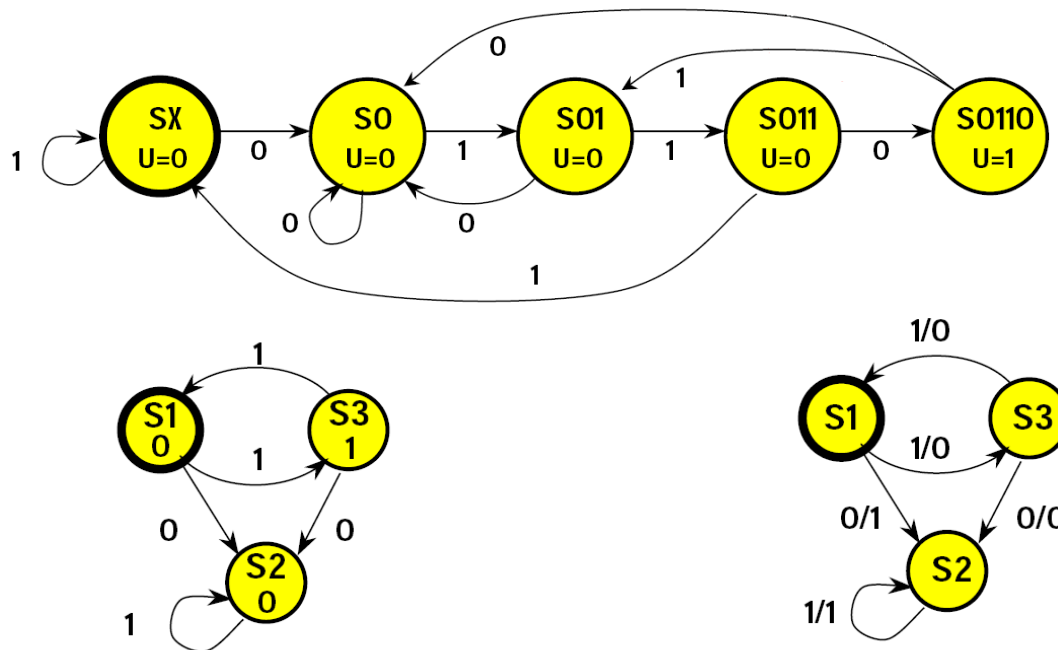




Máquina de Estados Finitos - MEF

- Diagrama de Estados
- Cada nó do diagrama representa um estado
- A transição entre estados é indicada por arcos
- As condições de disparo de uma transição são associadas aos arcos
- Cada estado corresponde a um ciclo de relógio

State Transition Diagram



MOORE Machine:
Outputs on States

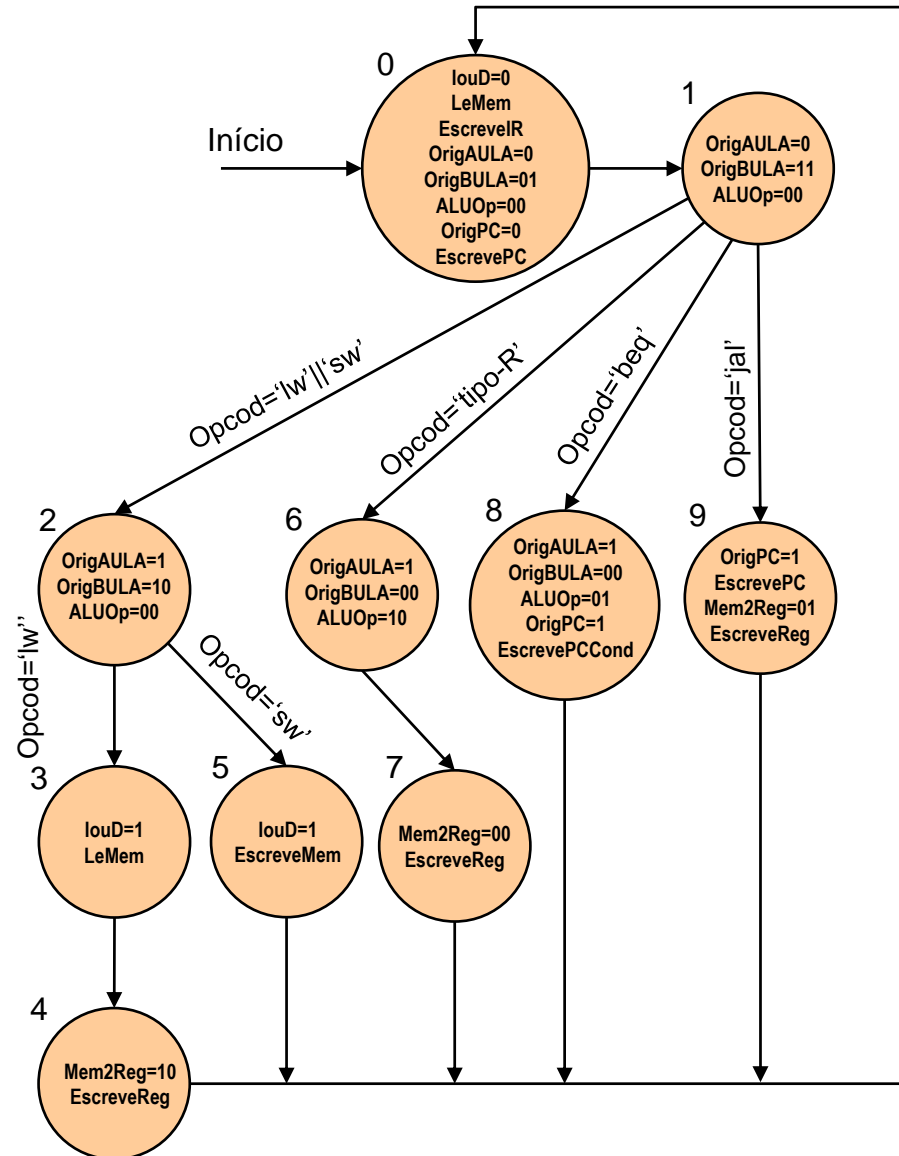
MEALY Machine:
Outputs on Transitions

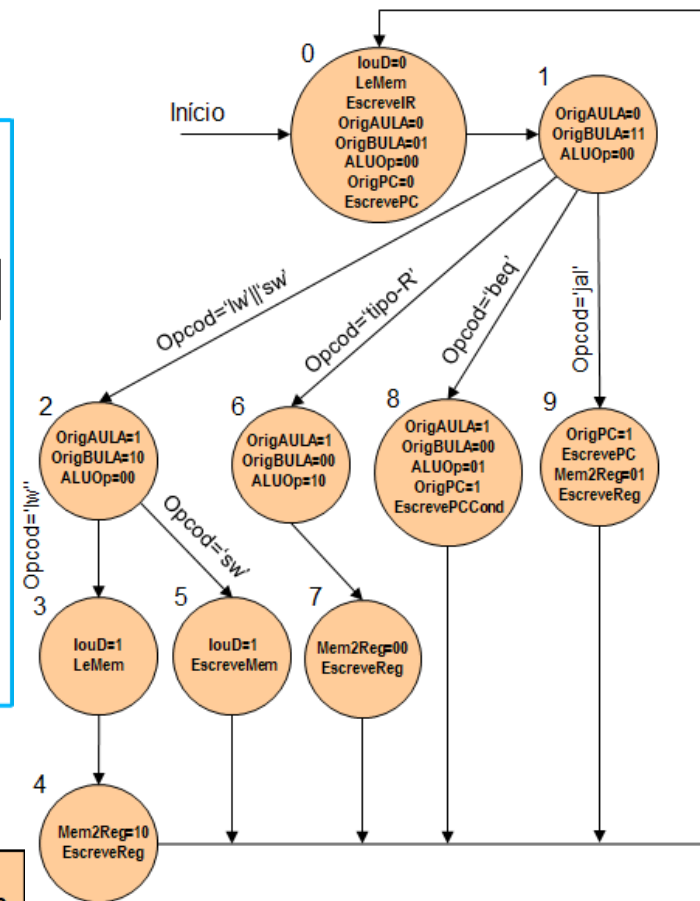
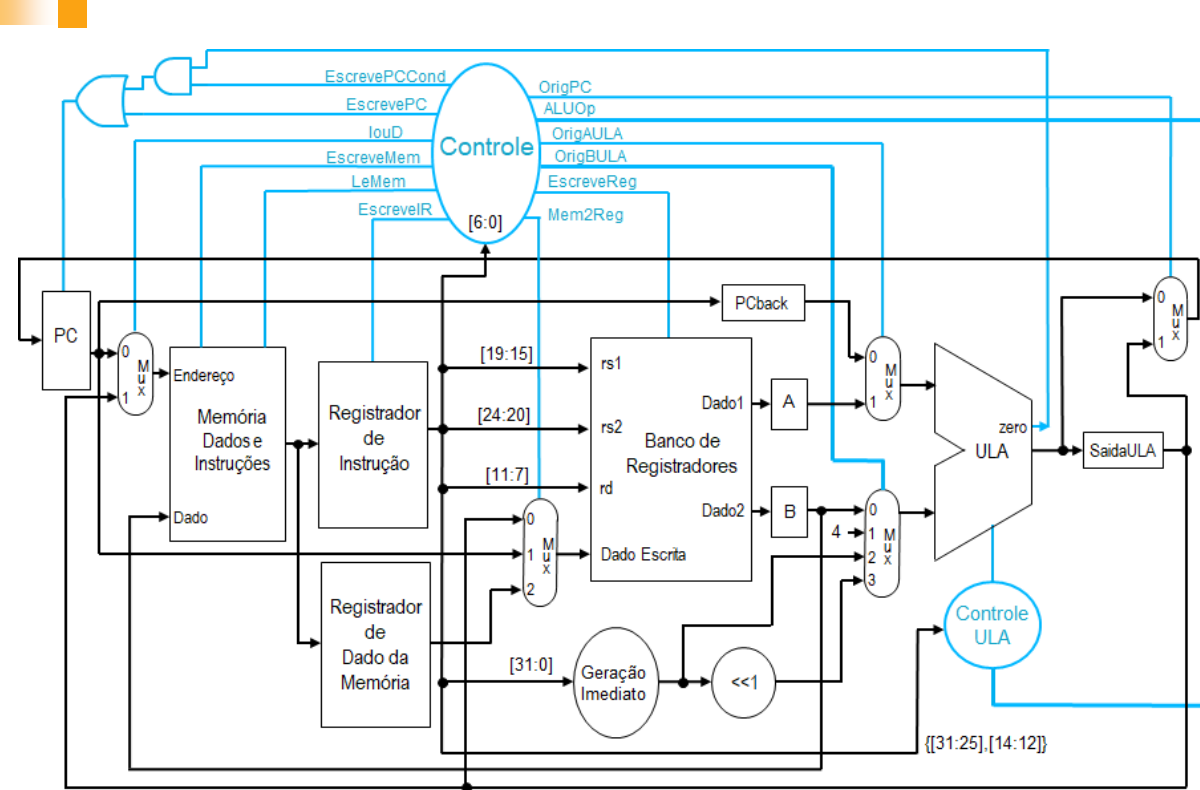


MEF do Controle do RISC-V Multiciclo

Análise do controle para toda a ISA implementada:

- 1) Busca da Instrução
- 2) Decodificação
- 3) Execução
- 4) Acesso à Memória e Conclusão Tipo-R
- 5) Conclusão LW



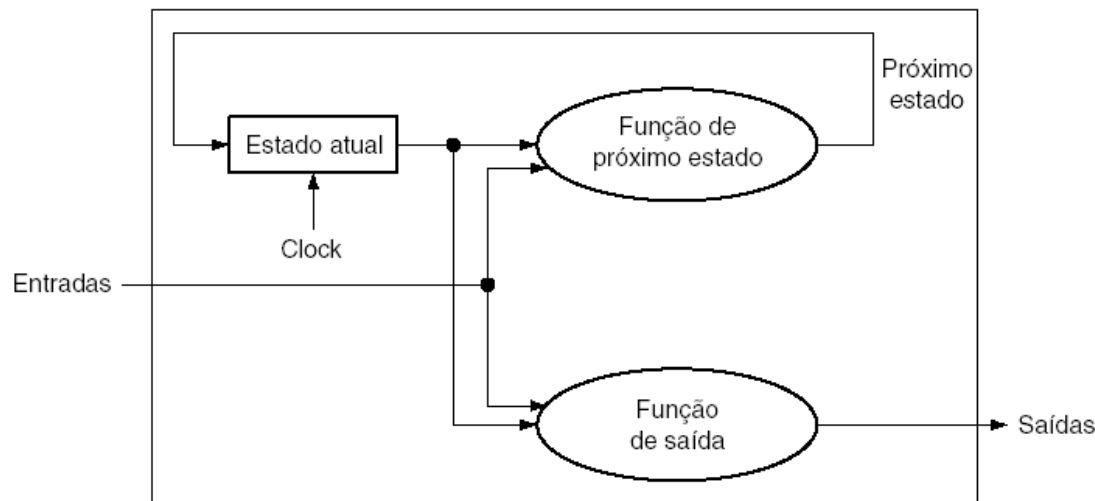


Etapa	Tipo-R	Acesso à Memória	Desvios Condicionais	Desvios Incondicionais
Busca da Instrução		IR<=Mem[PC] PCback<=PC PC<=PC+4		
Decodificação, Leitura dos registradores		A<=Reg[IR[19:15]] B<=Reg[IR[24:20]] SaidaULA<=PC+imm<<1		
Execução, cálculo do endereço	SaidaULA<=A op B	SaidaULA<=A+imm	Se (A==B) PC<=SaidaULA	Reg[IR[11:7]]<=PC+4 PC<=SaidaULA
Acesso à memória, conclusão tipo-R	Reg[IR[11:7]]<=SaidaULA	Load: MDR<=Mem[SaidaULA] Store: Mem[SaidaULA]<=B		
Conclusão lw		Load: Reg[IR[11:7]]<=MDR		



Máquinas de Estados Finitos - Implementação

- Conjunto de estados
- Função de próximo estado: Determinada pelo estado atual e entrada
- Saída: Determinada pelo estado atual (Moore) e possivelmente pela entrada (Mealy)

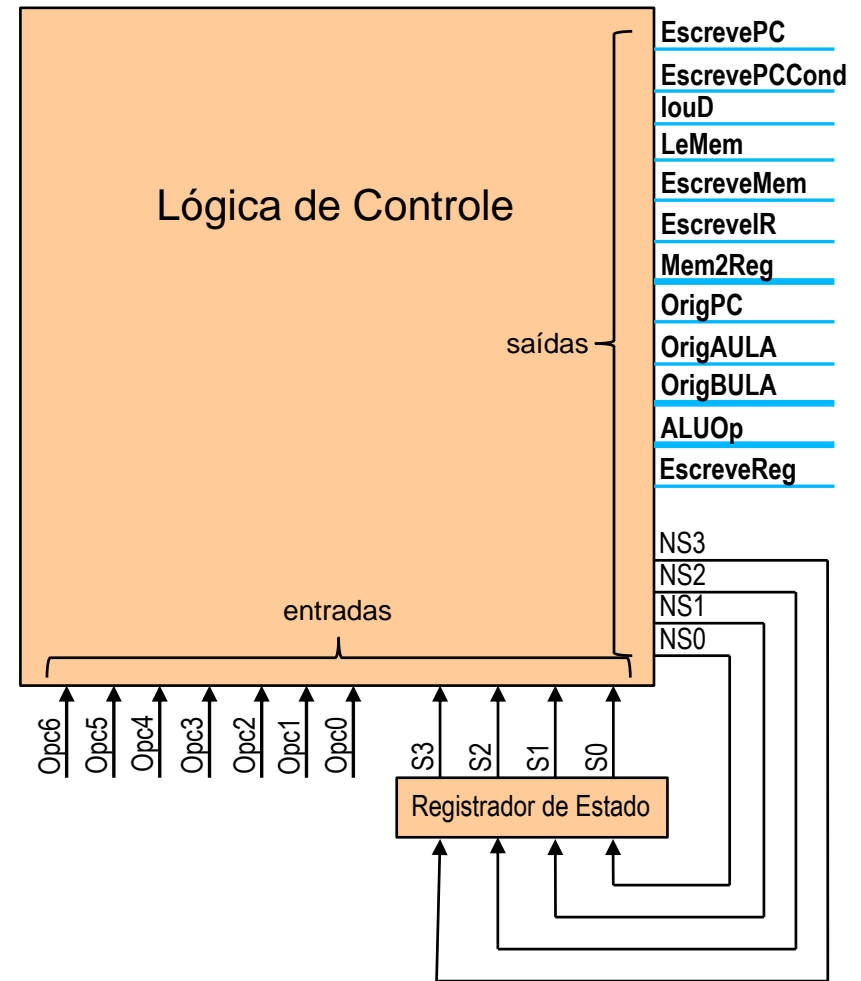
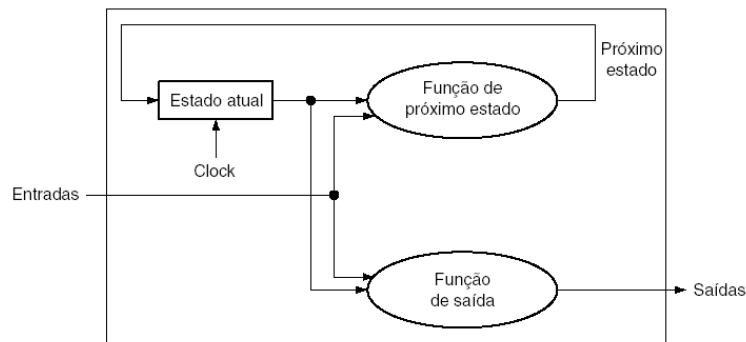




Controle do RISC-V com MEF

■ estrutura da máquina de estados:

- lógica de saída
- lógica de transição
- registrador de estado
- entradas externas (código da instrução)





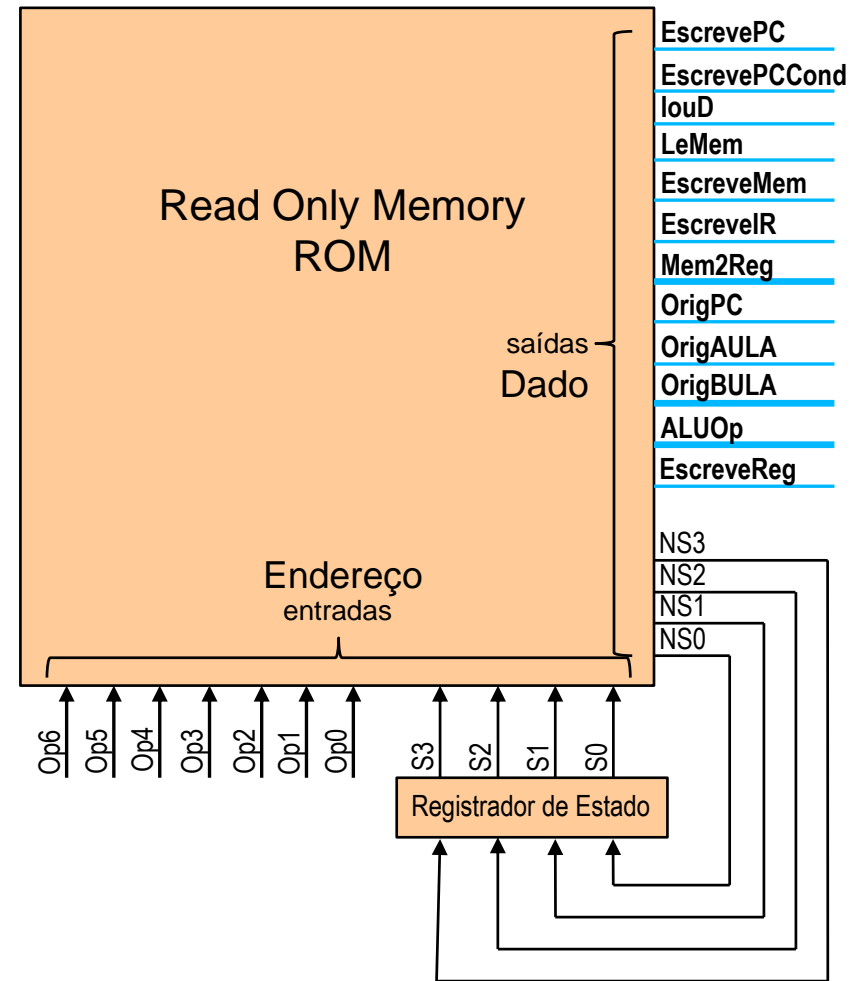
Controle com MEF

- Implementação com ROM
 - Simples!
- Tamanho da memória
 - 11 bits endereço:
2048 posições de memória
 - 18 bits de dados

Logo ROM de 36kibits

Quantas posições de memória
são realmente utilizadas?

- Porém, ineficiente





Exemplo do projeto lógico para cada saída na forma soma de produtos

- EscrevePC: Acionado nos estados 0 ou 9

S3	S2	S1	S0
0	0	0	0
1	0	0	1

$$\text{EscrevePC} = \bar{s}_3 \cdot \bar{s}_2 \cdot \bar{s}_1 \cdot \bar{s}_0 + s_3 \cdot \bar{s}_2 \cdot \bar{s}_1 \cdot s_0$$

- NS₀: Acionado nos estados 0, 2, 6 ou 1 (caso opcode=jal)

Op6	Op5	Op4	Op3	Op2	Op1	Op0	S3	S2	S1	S0
x	x	x	x	x	x	x	0	0	0	0
x	x	x	x	x	x	x	0	0	1	0
x	x	x	x	x	x	x	0	1	1	0
1	1	0	1	1	1	1	0	0	0	1

$$NS_0 = \bar{s}_3 \cdot \bar{s}_2 \cdot \bar{s}_1 \cdot \bar{s}_0 + \bar{s}_3 \cdot \bar{s}_2 \cdot s_1 \bar{s}_0 + \bar{s}_3 \cdot s_2 \cdot s_1 \bar{s}_0 + Op_6 \cdot Op_5 \cdot \overline{Op_4} \cdot Op_3 \cdot Op_2 \cdot Op_1 \cdot Op_0 \cdot \bar{s}_3 \cdot \bar{s}_2 \cdot \bar{s}_1 \cdot s_0$$



Controle com MEF

■ Implementação com PLA (Programmable Logic Array)

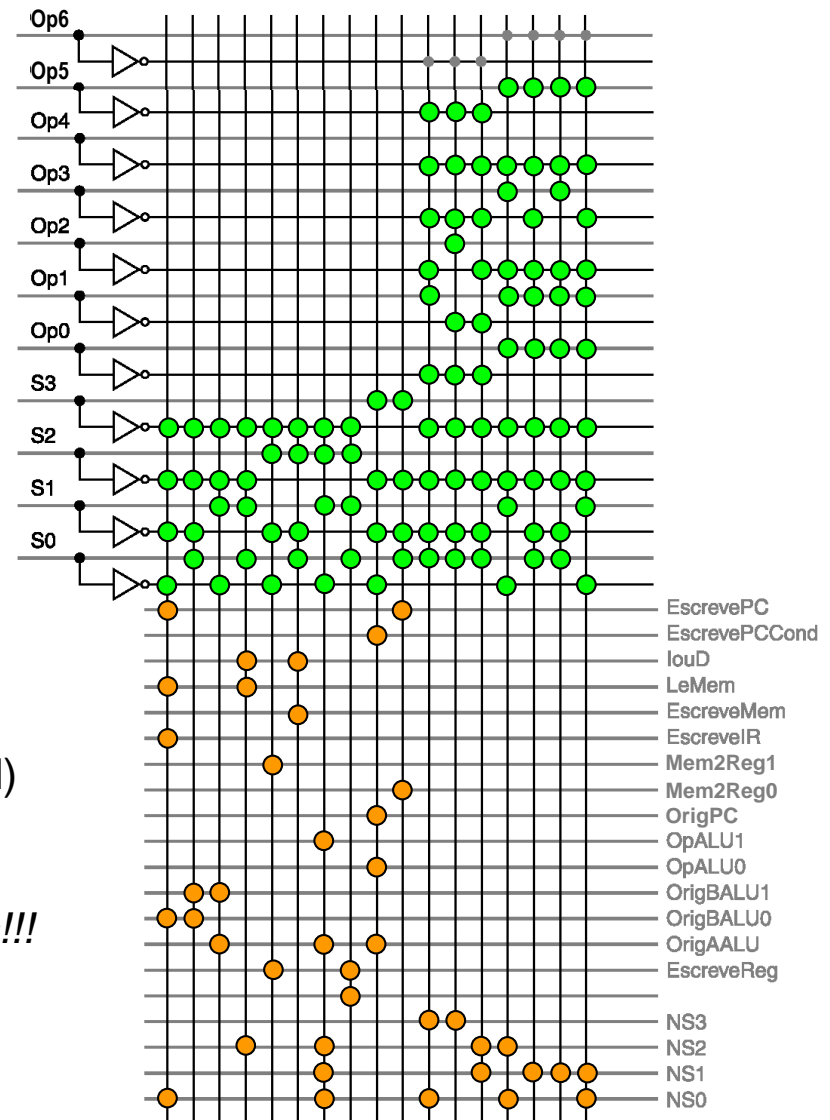
■ Mais eficiente:

- Pode compartilhar termos de produtos
- Apenas entradas que possuem saídas ativas
- Pode considerar *don't cares*

Tamanho=(Entradas×N.Prod.)+(Saídas×N.Prod)

Tamanho: (11×17)+(19×17)=510 células

Obs.: Precisa refazer a colocação das bolinhas!!!





Microprogramação

Problemas da MEF:

- O projeto da parte de controle através de diagramas de transição de estados pode rapidamente se tornar inviável se o número de estados for muito grande
- MEF's de processadores complexos (x86 e x64) podem ter milhares de estados

Uma alternativa para projeto é seguir um processo semelhante à programação



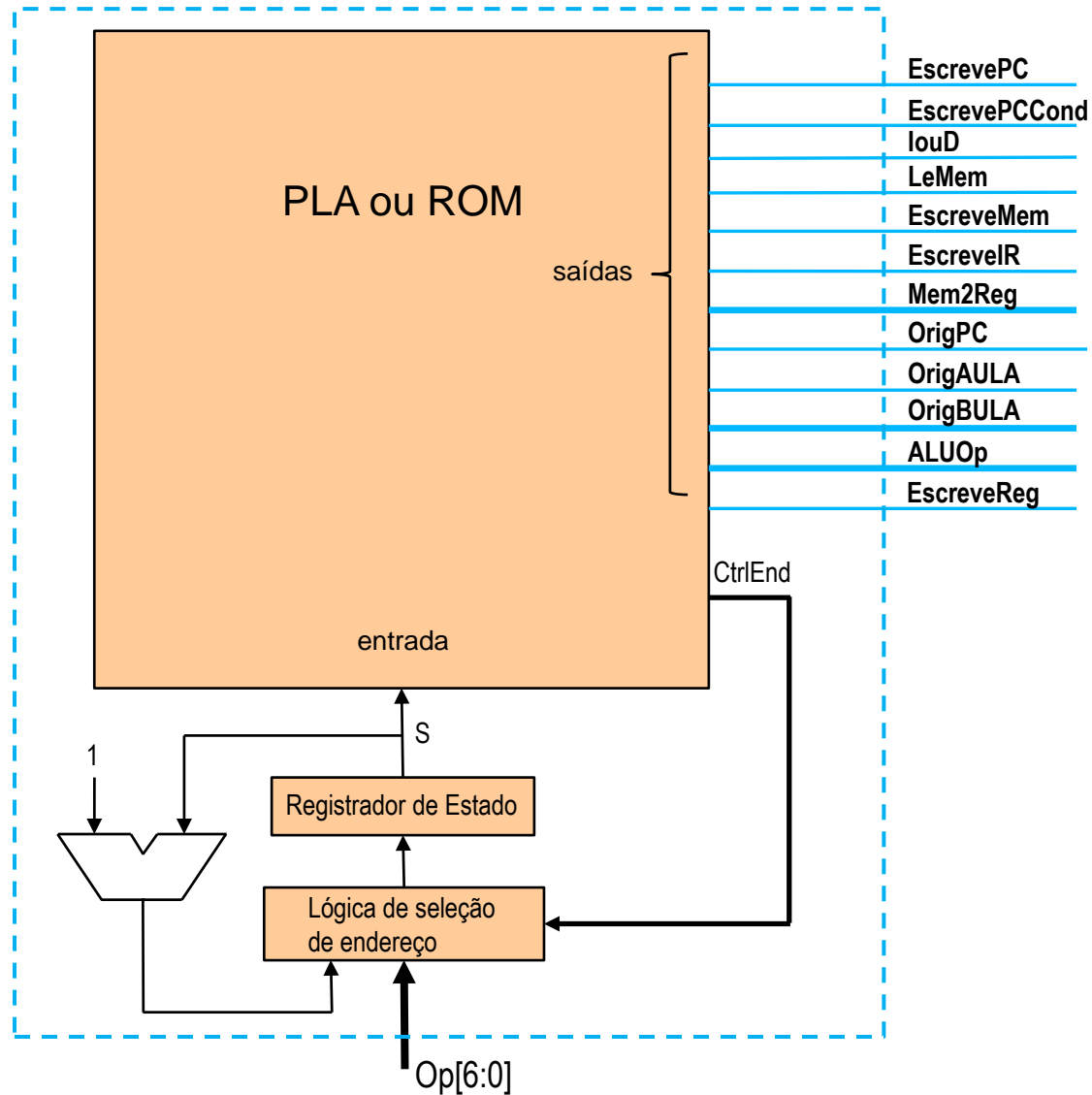
Microprogramação

- Uma **microinstrução** é definida pelos valores dos sinais de controle que atuam na unidade operativa durante um estado da MEF (ESTADO)
- A execução de uma instrução do processador pode então ser realizada através de uma **sequência** de microinstruções (TRANSIÇÕES)
- O conjunto de microinstruções que implementa o controle de um processador é chamado de **microprograma** (DIAGRAMA DE ESTADOS)



Estrutura do Sequenciador

Unidade de Controle





Microprograma

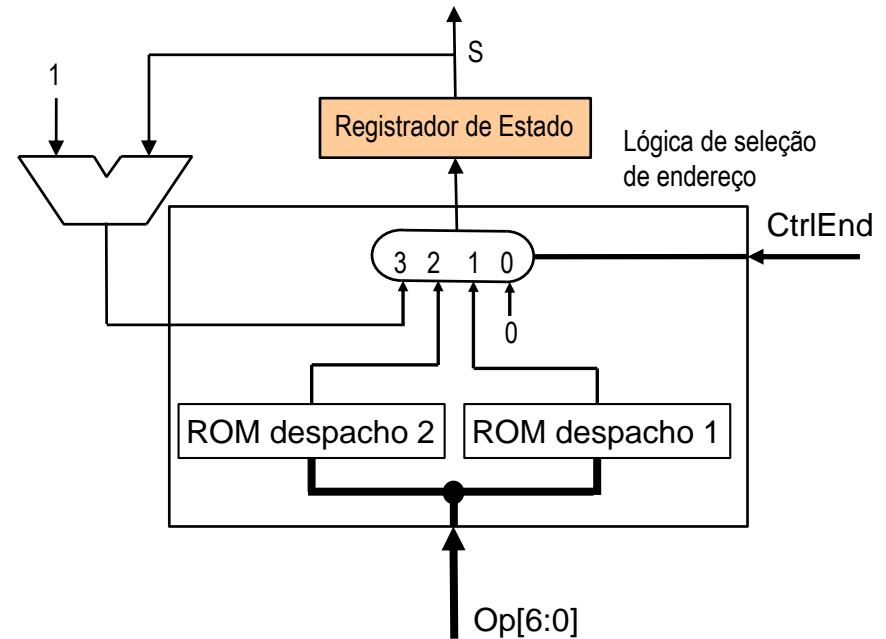
- O sequenciamento das microinstruções é realizado de forma similar a de um programa normal
 - microinstruções são usualmente executadas em sequência → correspondem aos caminhos no diagrama de estados.
 - em alguns casos, a sequência a ser seguida depende de informações externas (código da instrução, flags, exceções, interrupções). Nestes casos, são necessários mecanismos de desvio.



Sequenciamento para o subset do RISC-V

ROM de despacho 1		
Opcode	Instrução	Saída
0110011	Tipo-R	0110
1101111	jal	1001
1100011	beq	1000
0000011	lw	0010
0100011	sw	0010

ROM de despacho 2		
Opcode	Instrução	Saída
0000011	lw	0011
0100011	sw	0101

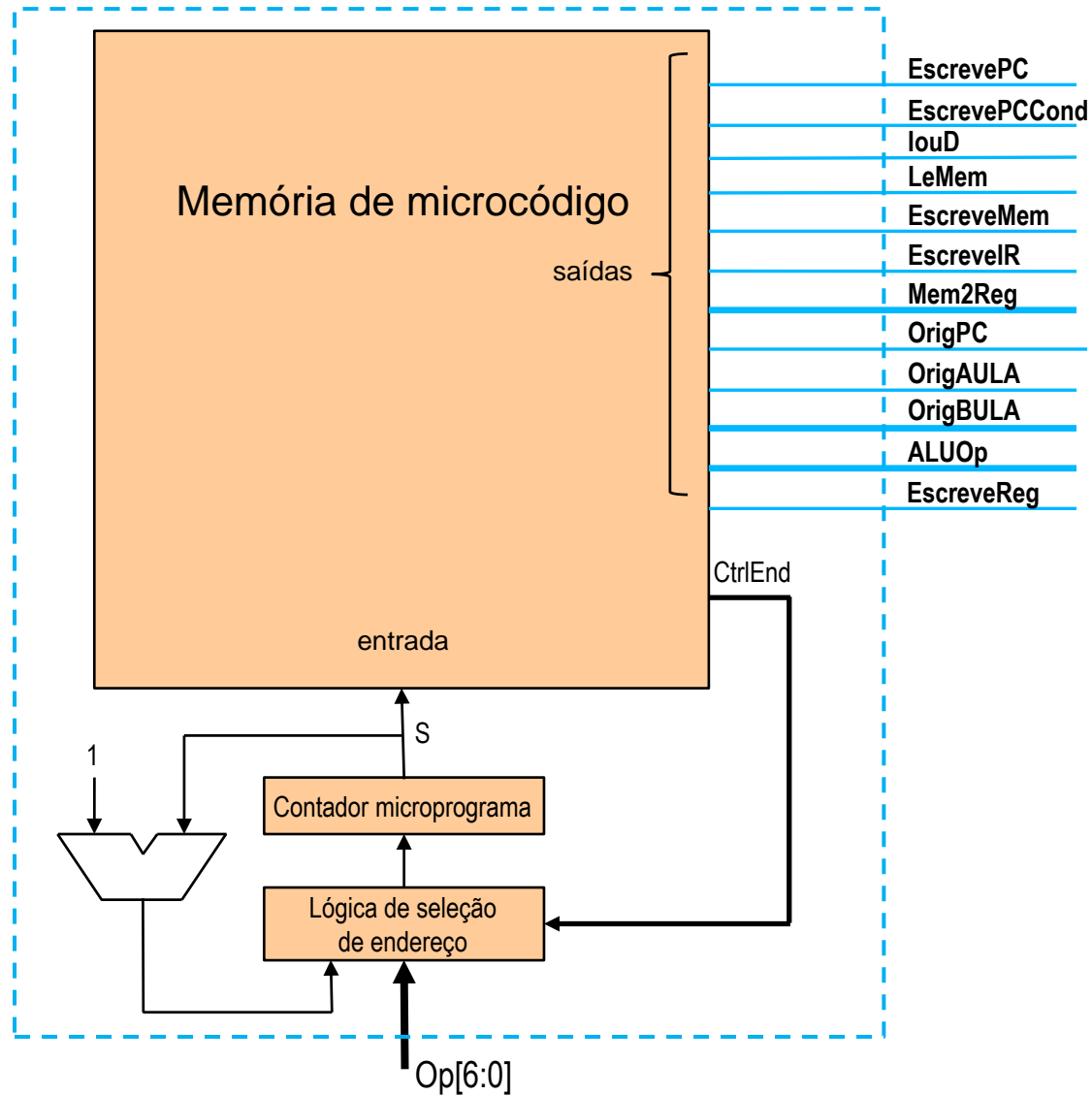


Número do Estado	Ação	CtrlEnd
0	Incrementa	3
1	ROM de despacho 1	1
2	ROM de despacho 2	2
3	Incrementa	3
4	Volta ao início	0
5	Volta ao início	0
6	Incrementa	3
7	Volta ao início	0
8	Volta ao início	0
9	Volta ao início	0



Sequenciador Microcodificado

Unidade de
Controle





Formato da Microinstrução

- A microinstrução é dividida em campos que atuam sobre conjuntos de elementos da unidade operativa
- Os campos são escolhidos de acordo com sua finalidade. O controle da ULA, por exemplo, é associado a um campo
- O microprograma é usualmente implementado em ROM, PLA, EEPROM, FLASH, etc., onde cada microinstrução tem seu próprio endereço



Função dos campos das Microinstruções

Nome do Campo	Função do Campo
Ctrl da ULA	Especifica a operação da ULA no ciclo de <i>clock</i> . Resultado é sempre escrito no registrador SaidaULA
Origem 1	Especifica o primeiro operando da ULA
Origem 2	Especifica o segundo operando da ULA
Ctrl Banco Regs	Especifica leitura ou escrita no Banco de Registradores, e a origem do valor de escrita
Memória	Especifica leitura ou escrita na Memória.
Ctrl escrita PC	Especifica a origem do PC
Sequenciação	Especifica com atingir a próxima microinstrução



MicroISA

Nome do Campo	Valor	Sinais Ativos	Comentário
Ctrl ULA	Add	ALUOp=00	ULA faz uma soma
	Sub	ALUOp=01	ULA faz uma subtração
	Funct	ALUOp=10	O campo funct define a operação
Orig1	PC	OrigAULA=0	Primeiro operando é o registrador PC
	A	OrigAULA=1	Primeiro operando vem do Banco de Registradores
Orig2	B	OrigBULA=00	Segundo operando vem do Banco de Registradores
	4	OrigBULA=01	Segundo operando é o valor 4
	Imm	OrigBULA=10	Segundo operando vem da unidade Geração de Imediato
	ShiftImm	OrigBULA=11	Segundo operando é o imediato deslocado 1 bit (x2)
Ctrl BR	Read		Le os dois registradores definidos nos campos rs1 e rs2
	WriteULA	Mem2Reg=0 EscreveReg=1	Escreve em rd o valor calculado pela ULA
	WriteMem	Mem2Reg=1 EscreveReg=1	Escreve em rd o valor lido da Memória
Ctrl Mem	ReadInstr	louD=0 LeMem=1 EscreveIR=1	Lê uma instrução da memória
	ReadData	louD=1 LeMem=1	Lê um dado da memória
	WriteData	louD=1 EscreveMem	Escreve um dado na memória
Ctrl PC	PC+4	OrigPC=0 EscrevePC=1	Escreve PC+4 no PC
	BranchAddress	OrigPC=1 EscrevePCCond=1	Desvio condicional
	JumpAddress	OrigPC=1 EscrevePC=1	Desvio incondicional
Seq	Incr	CtrlEnd=11	Incrementa o estado atual
	Fetch	CtrlEnd=00	Volta ao início
	Disp1	CtrlEnd=01	Usa a ROM de despacho1
	Disp2	CtrlEnd=10	Usa a ROM de despacho 2



Microprograma para a Unidade de Controle RISC-V

Label	Ctrl ULA	Orig1	Orig2	Ctrl BR	Mem	Ctrl PC	Seq	Endereço	microcódigo
Fetch:	Add	PC	4		ReadInstr	PC+4	Incr	0x00	0000101000101011
	Add	PC	ShiftImm	Read			Disp1	0x01	
Mem1:	Add	A	Imm				Disp2	0x02	
Lw2:					ReadData		Incr	0x03	
					WriteMem		Fetch	0x04	
Sw2:					WriteData		Fetch	0x05	
R-Type1:	Funct	A	B				Incr	0x06	
					WriteALU		Fetch	0x07	
Beq1:	Sub	A	B			BranchAddress	Fetch	0x08	
Jal1:						JumpAddress	Fetch	0x09	

ROM de despacho 1	
Endereço	Conteúdo
0110011	0110
1101111	1001
1100011	1000
0000011	0010
0100011	0010

ROM de despacho 2	
Endereço	Conteúdo
0000011	0011
0100011	0101

Obs.: O microcódigo depende da posição dos sinais nos campos!



Exercício

- Considerando o workload do compilador gcc, qual a CPI média do RISC-V multiciclo implementado?
 - Load: 22% (5 ciclos)
 - Store: 11% (4 ciclos)
 - Operações logico-aritméticas: 49% (4 ciclos)
 - Desvios Condicionais: 16% (3 ciclos)
 - Desvios Incondicionais: 2% (3 ciclos)

$$CPI = 0.22 \times 5 + 0.11 \times 4 + 0.49 \times 4 + 0.16 \times 3 + 0.02 \times 3 = 4.04$$