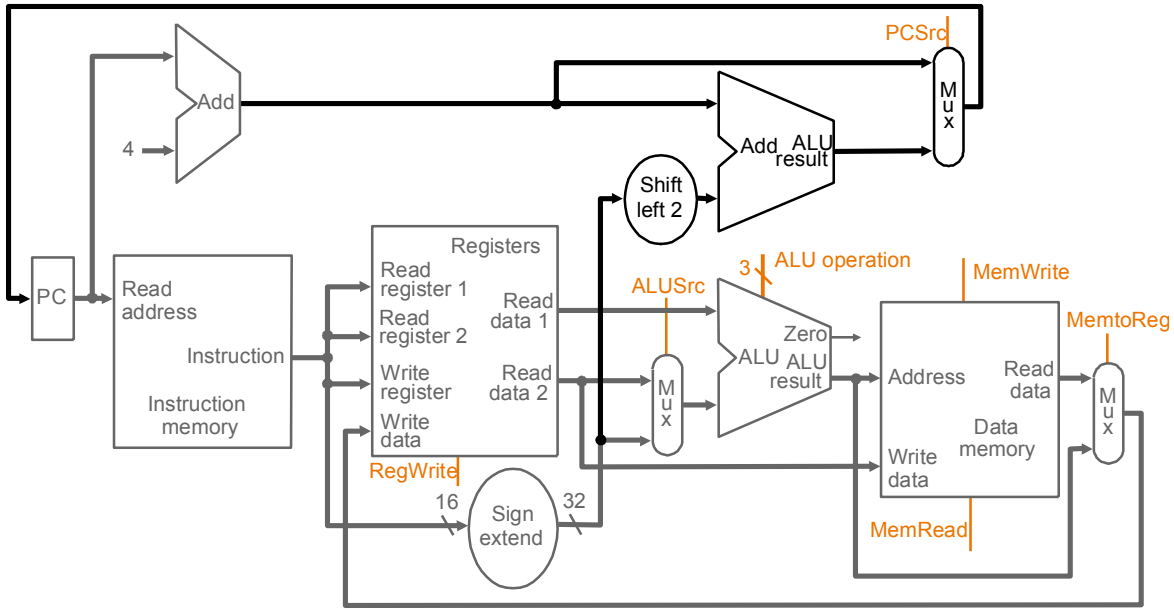
Organização de Computadores I
DCC006

Aula 7 – O Processador: Data Path e Controle (Continuação)

Prof. Omar Paranaiba Vilela Neto

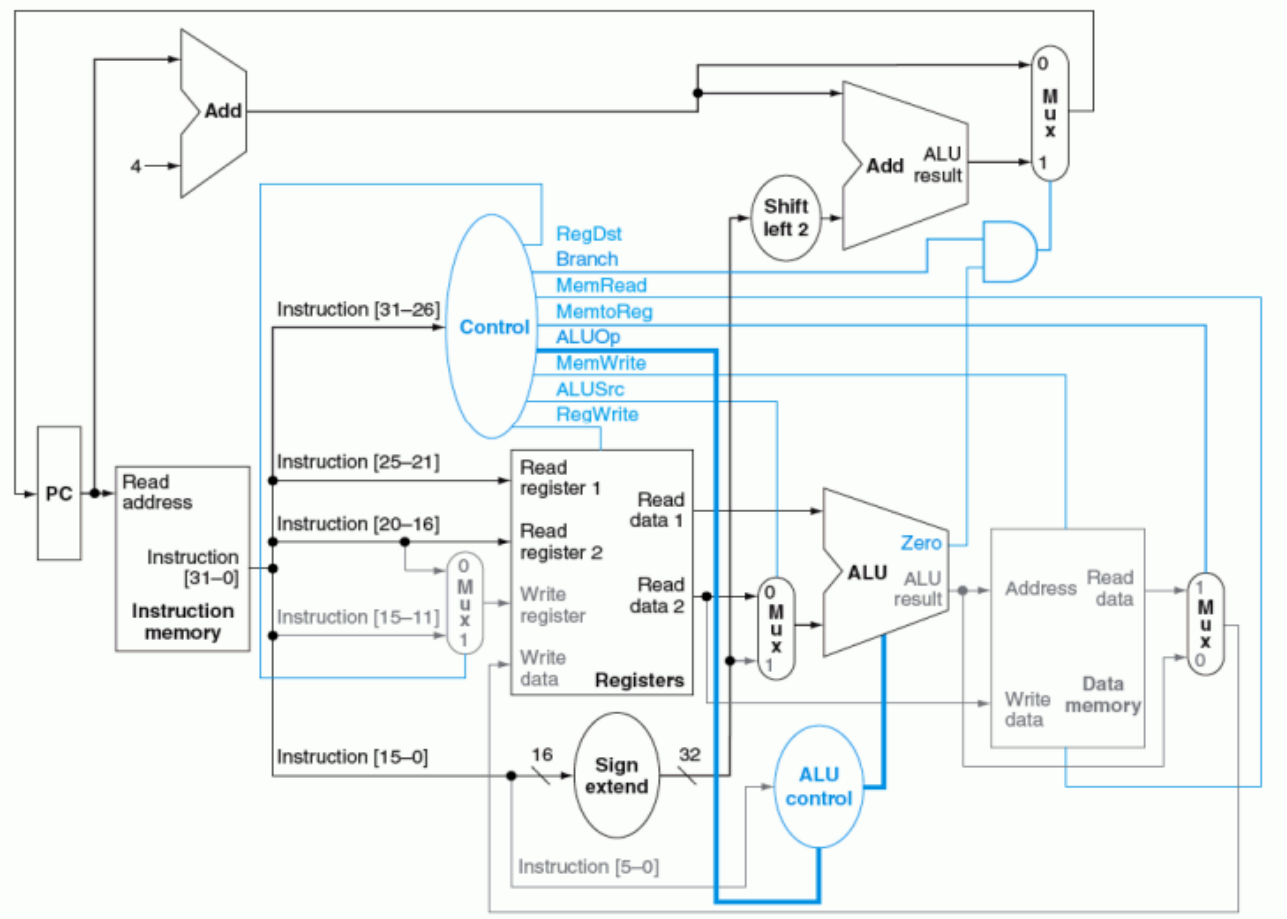
Relembrando aulas anteriores

Datapath sem controle - MIPS



Relembrando aulas anteriores

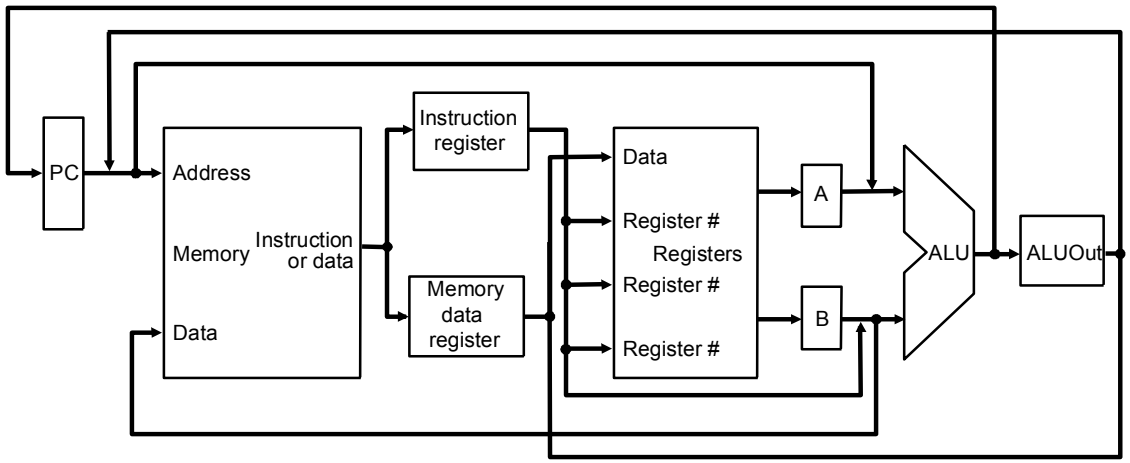
Datapath com controle - MIPS



Relembrando aulas anteriores

Problemas

- **Problemas do ciclo único:**
  - **Instruções complicadas como ponto flutuante, como fazer?**
  - **Gasto de área**
- **Uma solução:**
  - **usar o menor cycle time**
  - **Ter diferentes números de ciclos para instruções diferentes**
  - **datapath "multiciclo":**

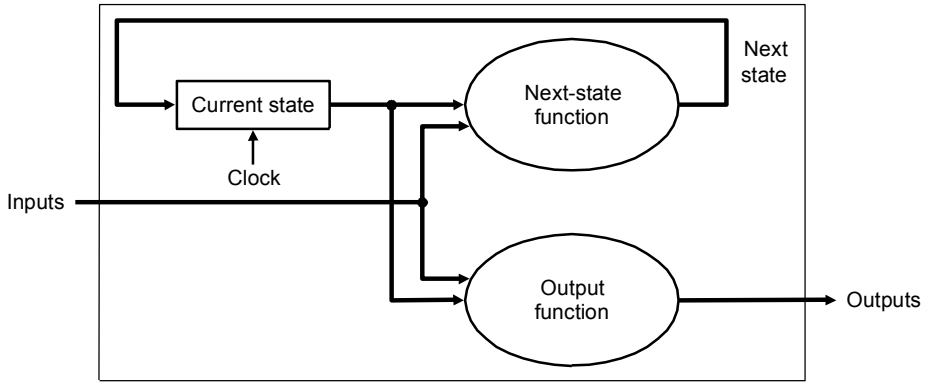


Abordagem Multiciclo

- Nós reusaremos as unidades funcionais
  - ALU usada para calcular endereço e incrementar PC
  - Memória usada para instruções e dados
- Nossos sinais de controle não serão determinados somente pelas instruções
- Controle deverá conhecer o estado corrente - Máquina de Estados

Revisão: Máquinas de Estados finitos

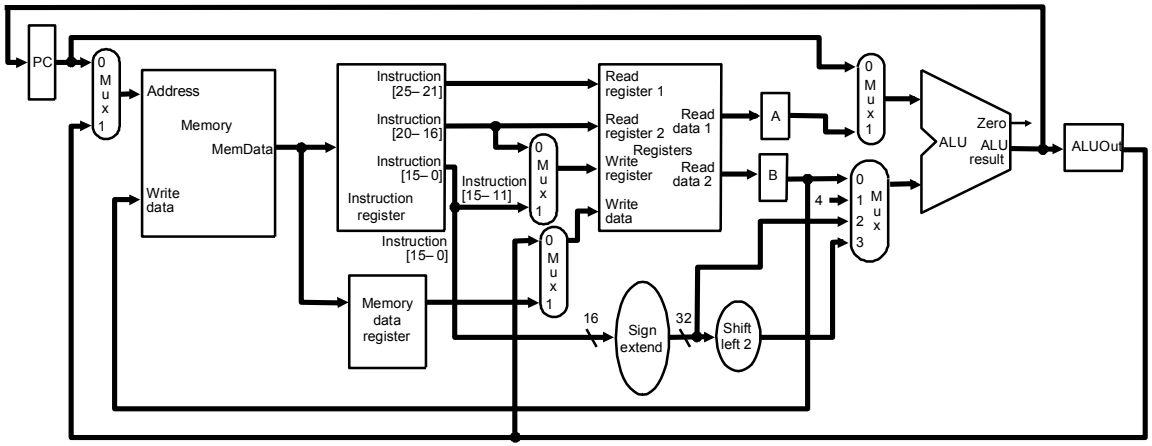
- **Maquinas de estados finitos:**
  - **Um conjunto de estados e**
  - **próximo estado (determinado pelo estado corrente e entradas)**
  - **saídas (determinadas pelo estado corrente e entradas)**



  - **Máquina de Moore (saídas baseadas somente no estado corrente)**

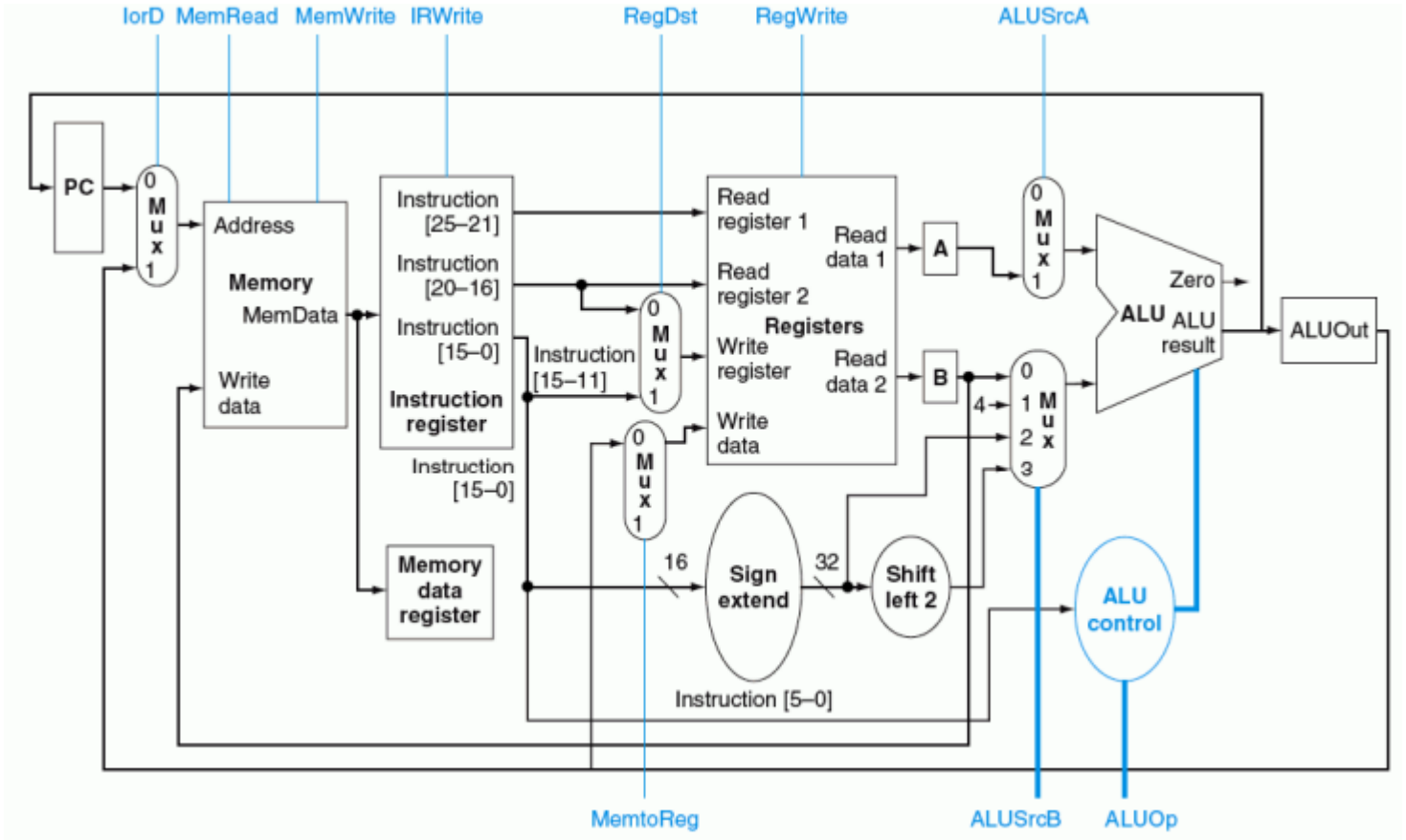
Abordagem Multiciclo

- **Dividir as instruções em passos**, cada passo em um ciclo
  - –**balanceamento** entre o trabalho a ser feito
  - –restringir em cada ciclo o **uso de só uma unidade funcional**
- **ao final de cada ciclo**
  - –**armazenar valores** a serem usados nos ciclos subsequentes
  - –**introduzir registradores** internos adicionais



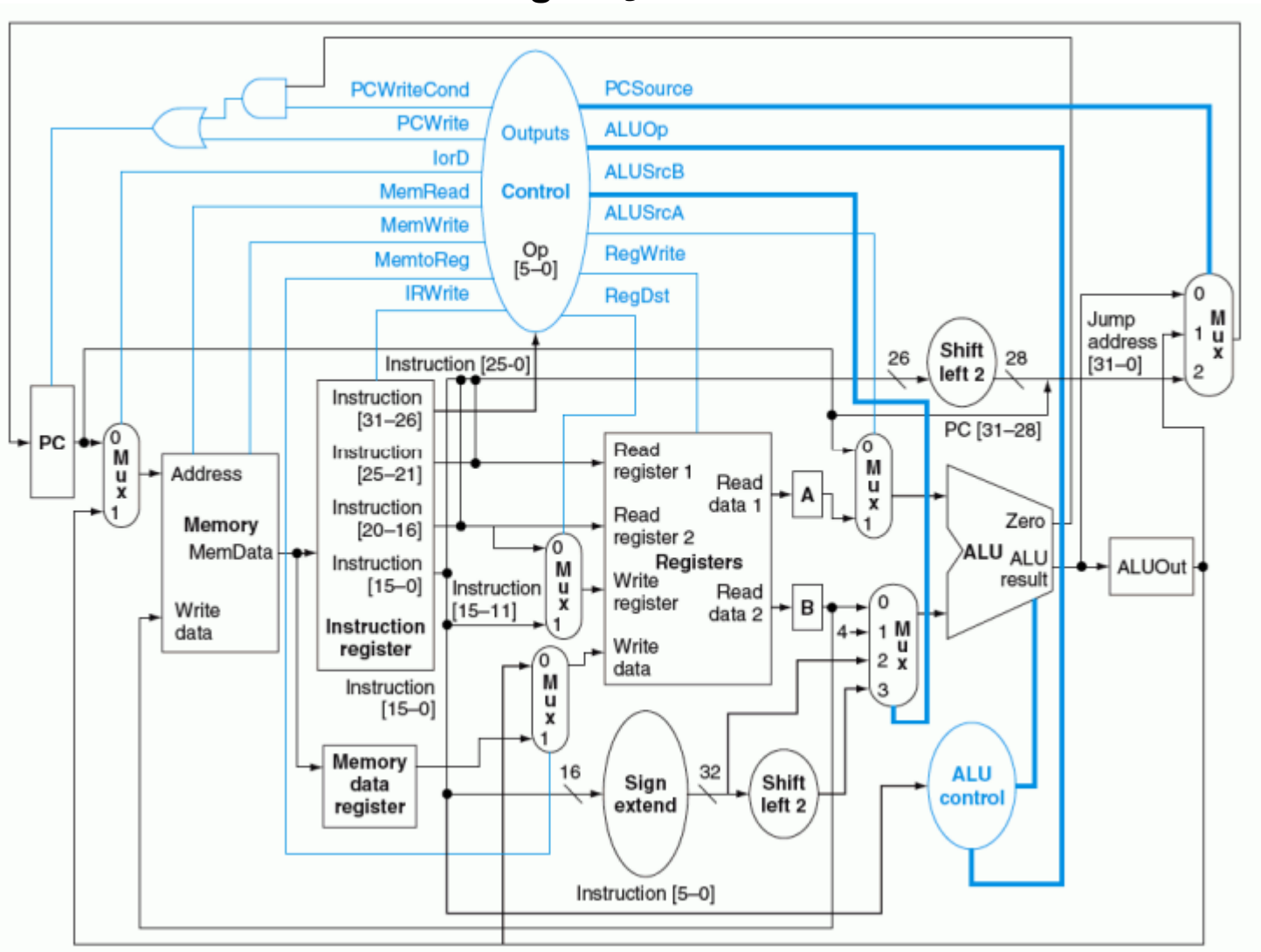
Abordagem Multiciclo

- **Precismos de uma nova configuração de controle**



Abordagem Multiciclo

- **Precismos de uma nova configuração de controle**



Cinco passos da Execução

- **Instruction Fetch**
- **Instruction Decode and Register Fetch**
- **Execution, Memory Address Computation, or Branch Completion**
- **Memory Access or R-type instruction completion**
- **Write-back step**

- ***INSTRUÇÕES USAM DE 3 A 5 CICLOS!***

Passo 1: Instruction Fetch

- **Usa o PC para buscar a instrução e coloca a instrução no registrador de instrução.**
- **Incrementa o PC de 4 e coloca o resultado de novo no PC.**
- **Pode ser sucintamente escrito usando "Register-Transfer Language" RTL**

```
IR = Memory[PC];
PC = PC + 4;
```

- ***Nós podemos descrever os sinais de controle?***

- ***Qual a vantagem de atualizar o PC agora?***

Passo 2: Instruction Decode and Register Fetch

- **Ler registradores rs e rt no caso de necessidade**
- **calcular o endereço do branch address no caso da instrução ser branch**
- **RTL:**

```
A = Reg[IR[25-21]];
B = Reg[IR[20-16]];
ALUOut = PC + (sign-extend(IR[15-0]) << 2);
```

- **Nós não ativamos linhas de controle baseados no tipo de instrução**
- **(No nosso controle nós estamos decodificando a instrução)**

Passo 3 (depende da instrução)

- **ALU executa uma das três funções, baseada no tipo da instrução**
- **Memory Reference:**
- `ALUOut = A + sign-extend(IR[15-0]);`
- **R-type:**
- `ALUOut = A op B;`
- **Branch:**
- `if (A==B) PC = ALUOut;`

Passo 4 (R-type ou memory-access)

- **Loads e stores acesso a memória**

```
MDR = Memory[ALUOut];
 or
Memory[ALUOut] = B;
```

- **R-type instrução terminada**

```
Reg[IR[15-11]] = ALUOut;
```

- ***A escrita é feita no final do ciclo***

Write-back

- `Reg[IR[20-16]]= MDR;`

O que dizer sobre outras instruções?

Resumo:

Step name	instructions	instructions	branches	jumps
Instruction fetch	IR = Memory[PC] PC = PC + 4			
Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut	PC = PC [31-28] II (IR[25-0]<<2)
Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Questões Simples

- **Quantos ciclos são necessários para executar este código?**

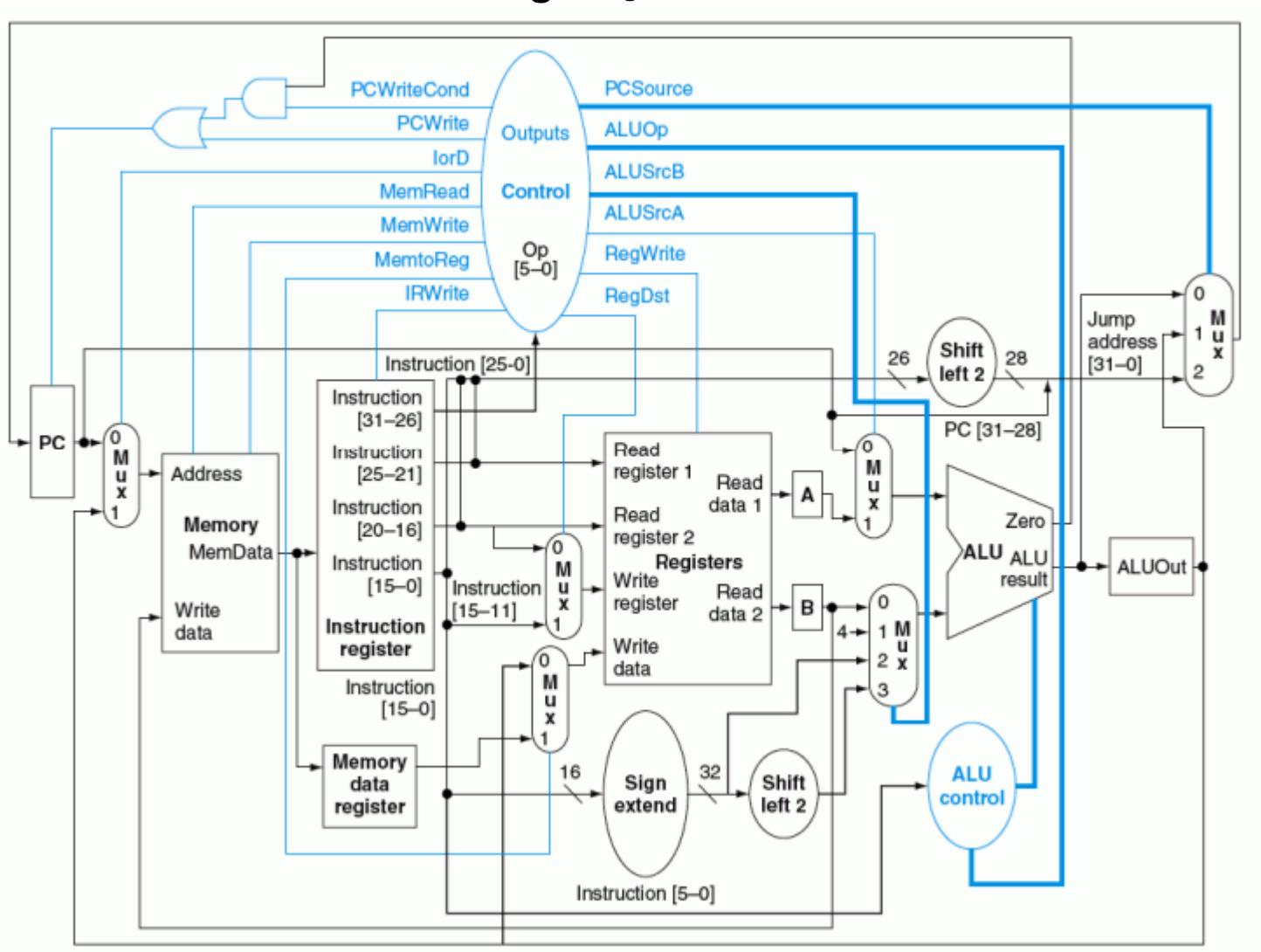
```
•   lw $t2, 0($t3)
•   lw $t3, 4($t3)
•   beq $t2, $t3, Label     #assume não
•   add $t5, $t2, $t3
•   sw $t5, 8($t3)
Label: ...
```

- **Para onde vamos durante a execução do 8º ciclo?**
- **Em qual ciclo a soma de `$t2` e `$t3` é executada?**

•

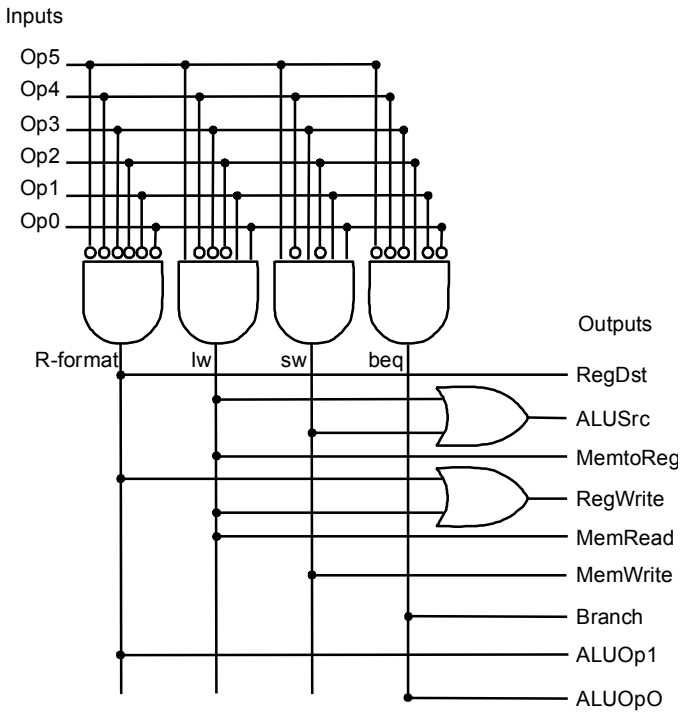
Abordagem Multiciclo

- **Precismos de uma nova configuração de controle**



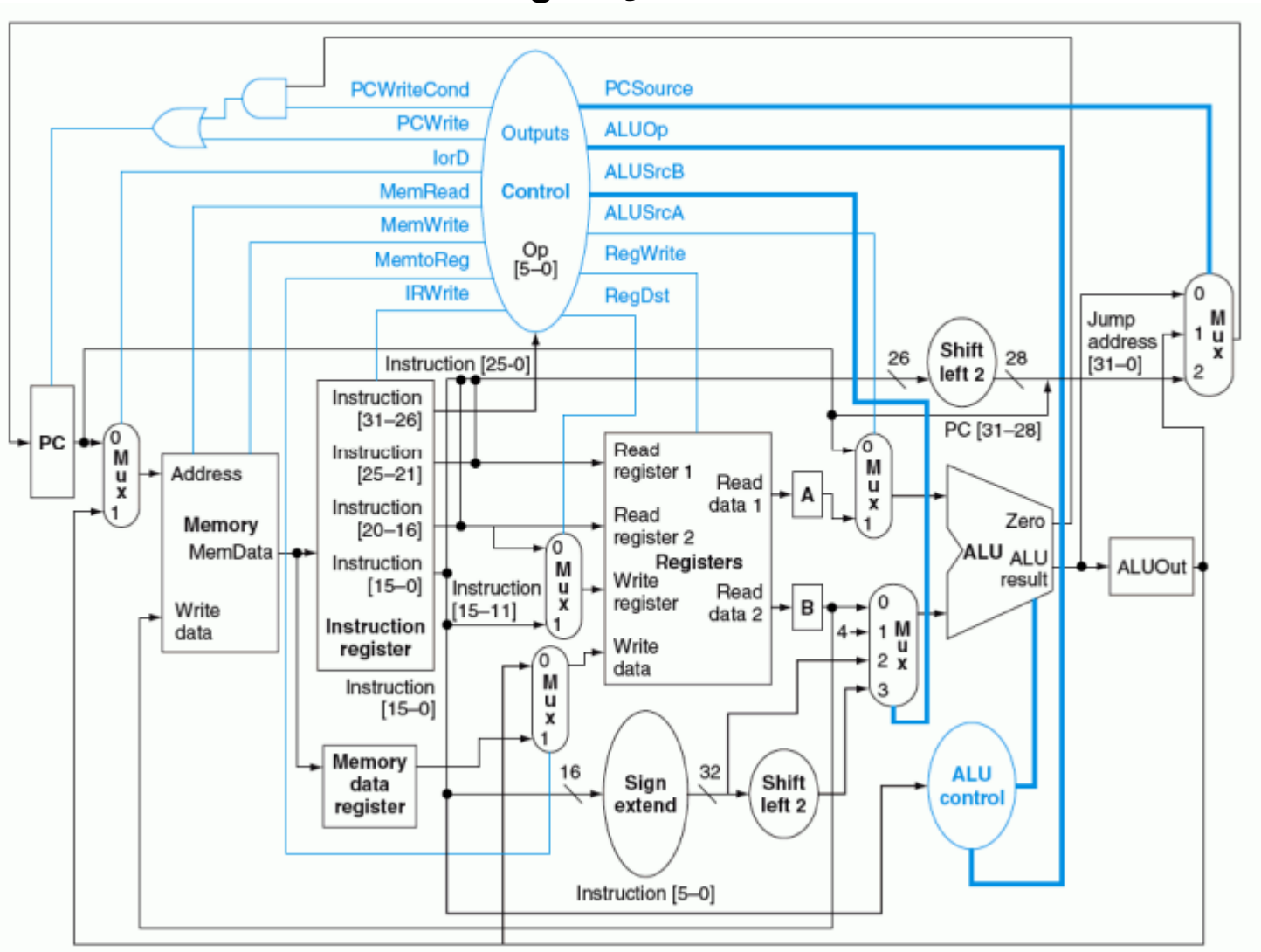
Controle do Ciclo Único

Tabela Verdade



Controle Multiciclo

- **Precismos de uma nova configuração de controle**

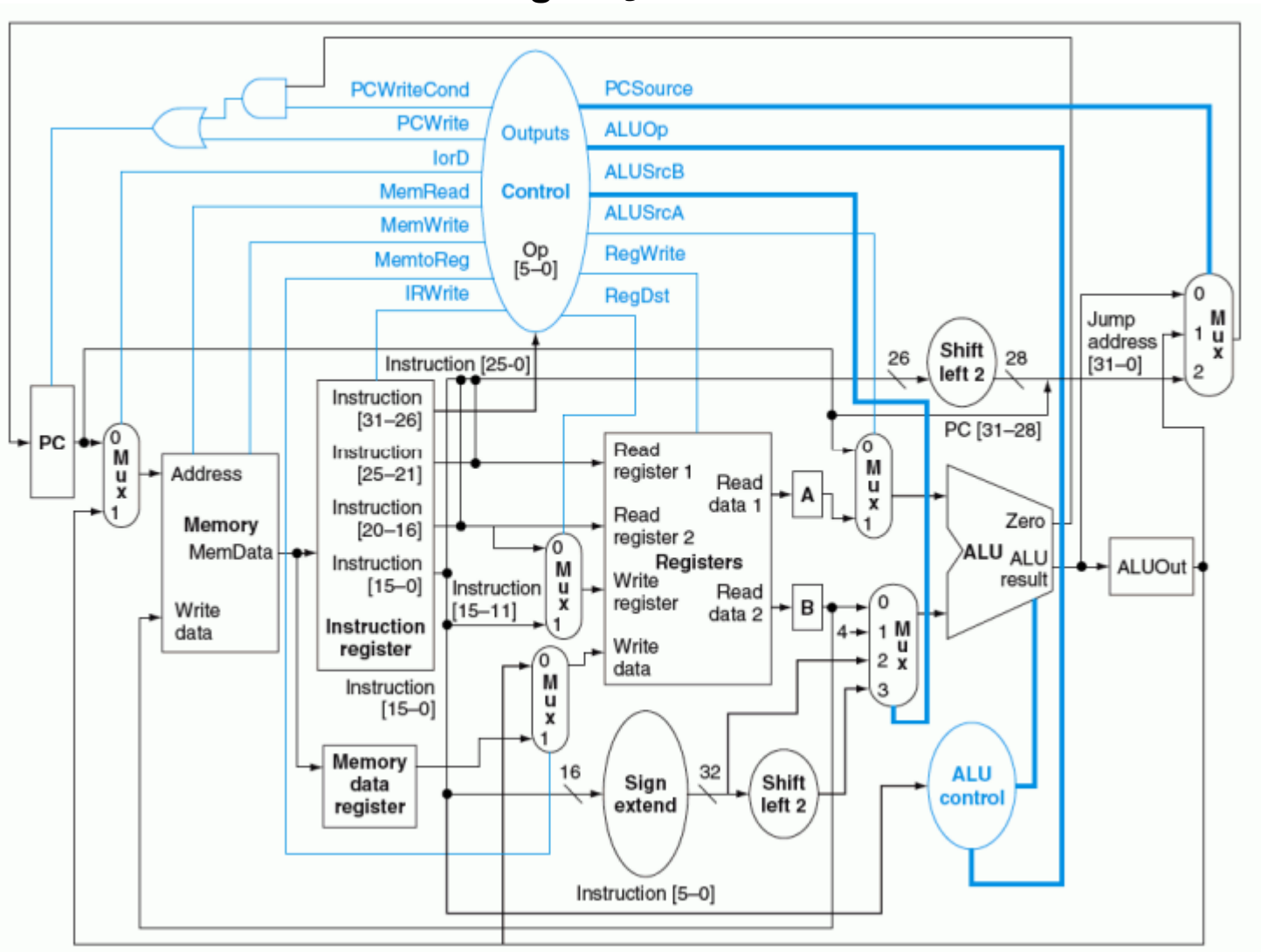


Controle Multiciclo

- Valores dos sinais de controle são dependentes de:
  - Qual instrução esta sendo executada
  - Qual passo do ciclo esta sendo processado
- Uso da informação especificada pela máquina de estados finitos
  - especificar a máquina de estados finitos, ou
  - usar microprogramação
- Implementação pode ser derivada de uma especificação

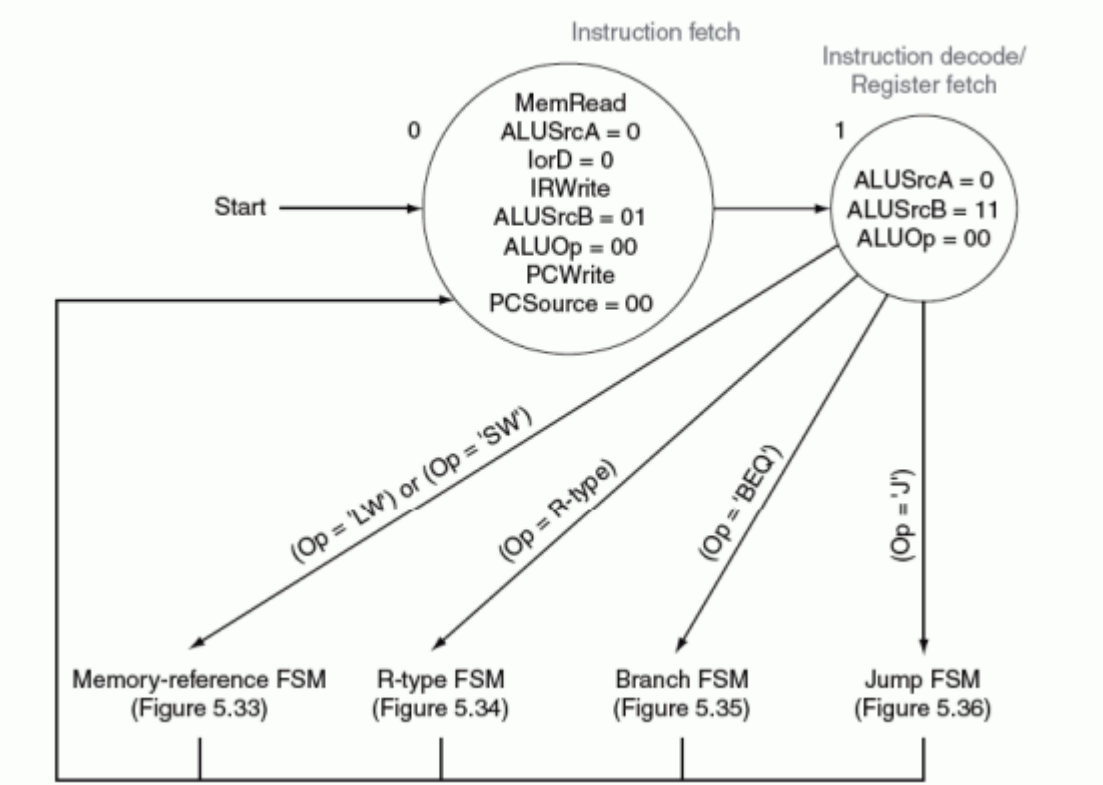
Controle Multiciclo

- **Precismos de uma nova configuração de controle**



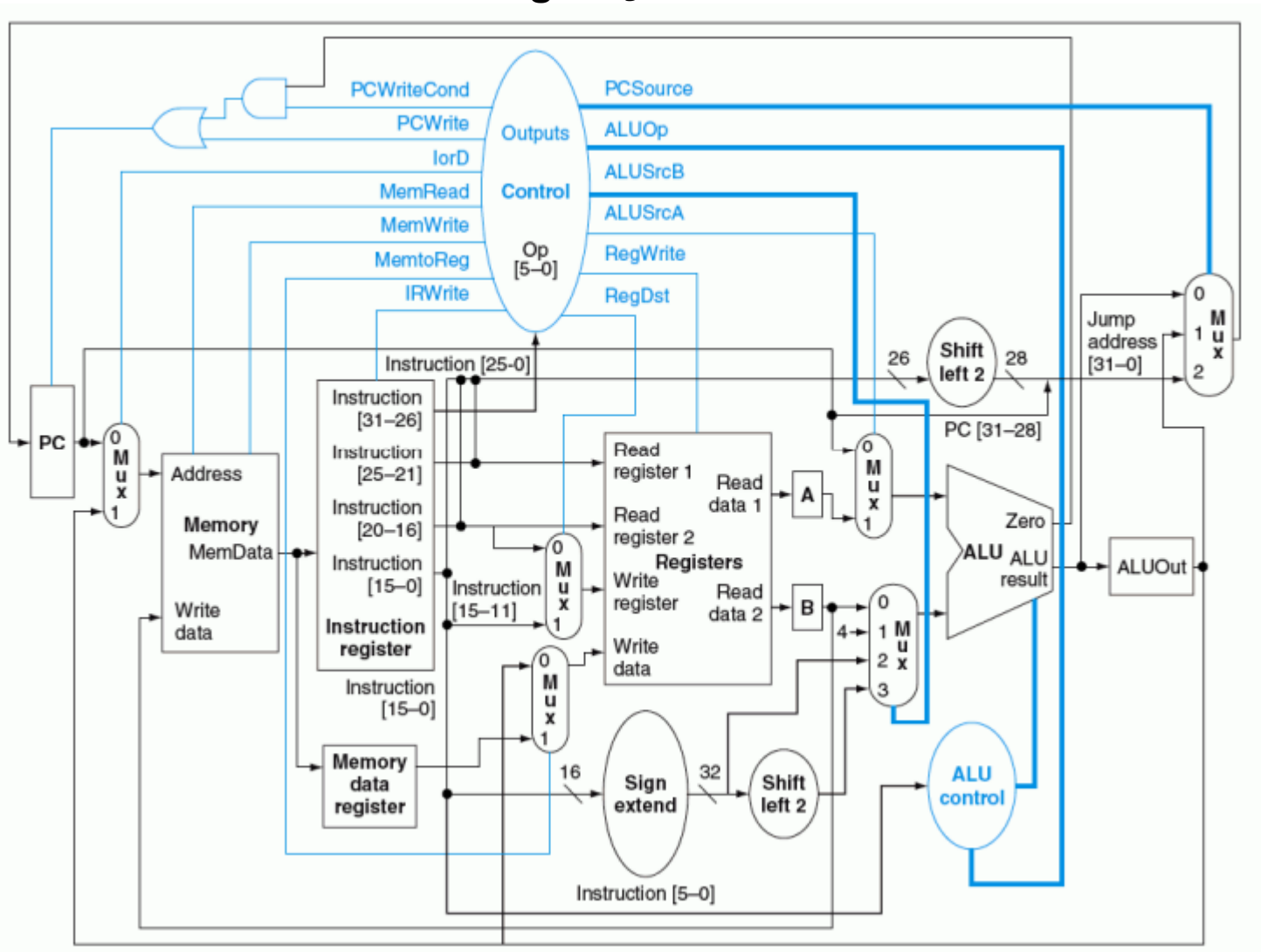
Máquina de Estados Finitos - Controle

Estados Iniciais



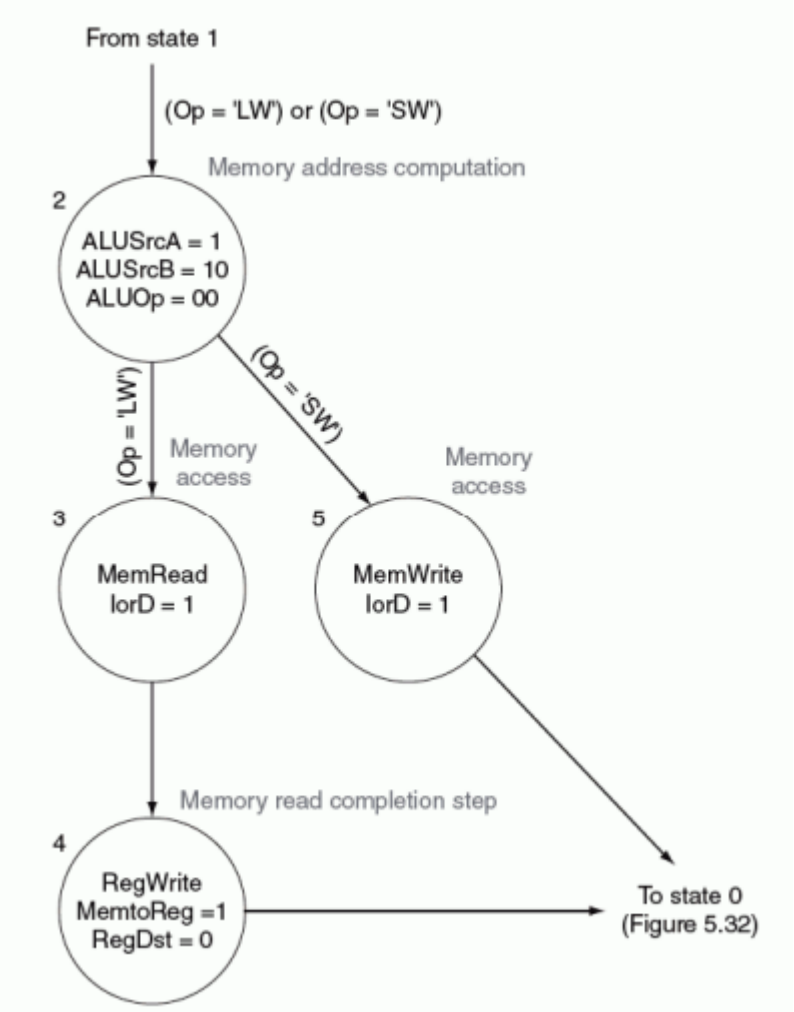
Controle Multiciclo

- **Precismos de uma nova configuração de controle**



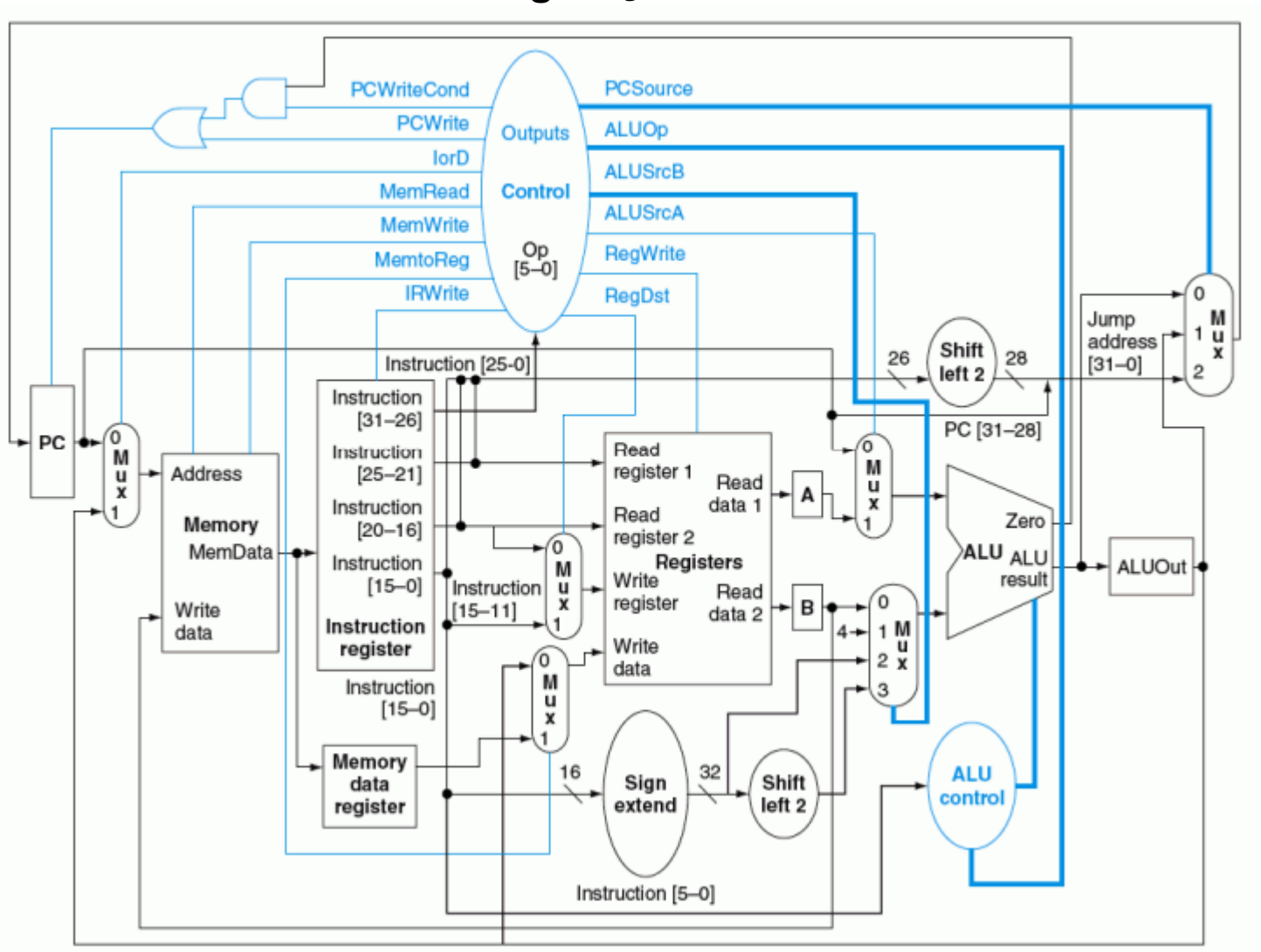
Máquina de Estados Finitos - Controle

Acesso à Memória



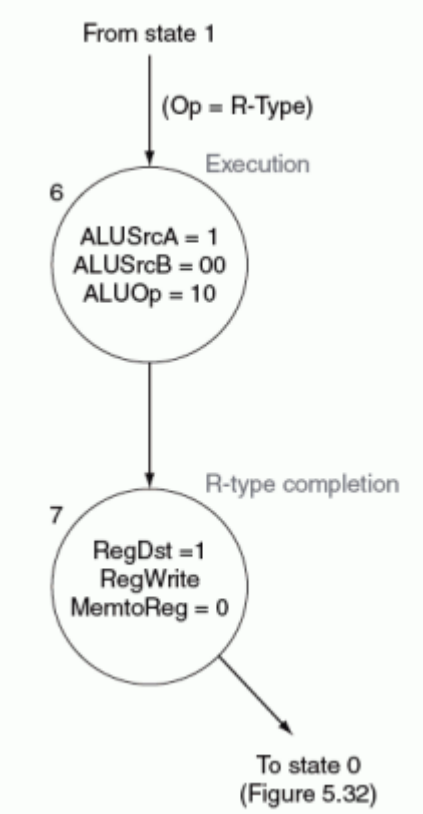
Controle Multiciclo

- **Precismos de uma nova configuração de controle**



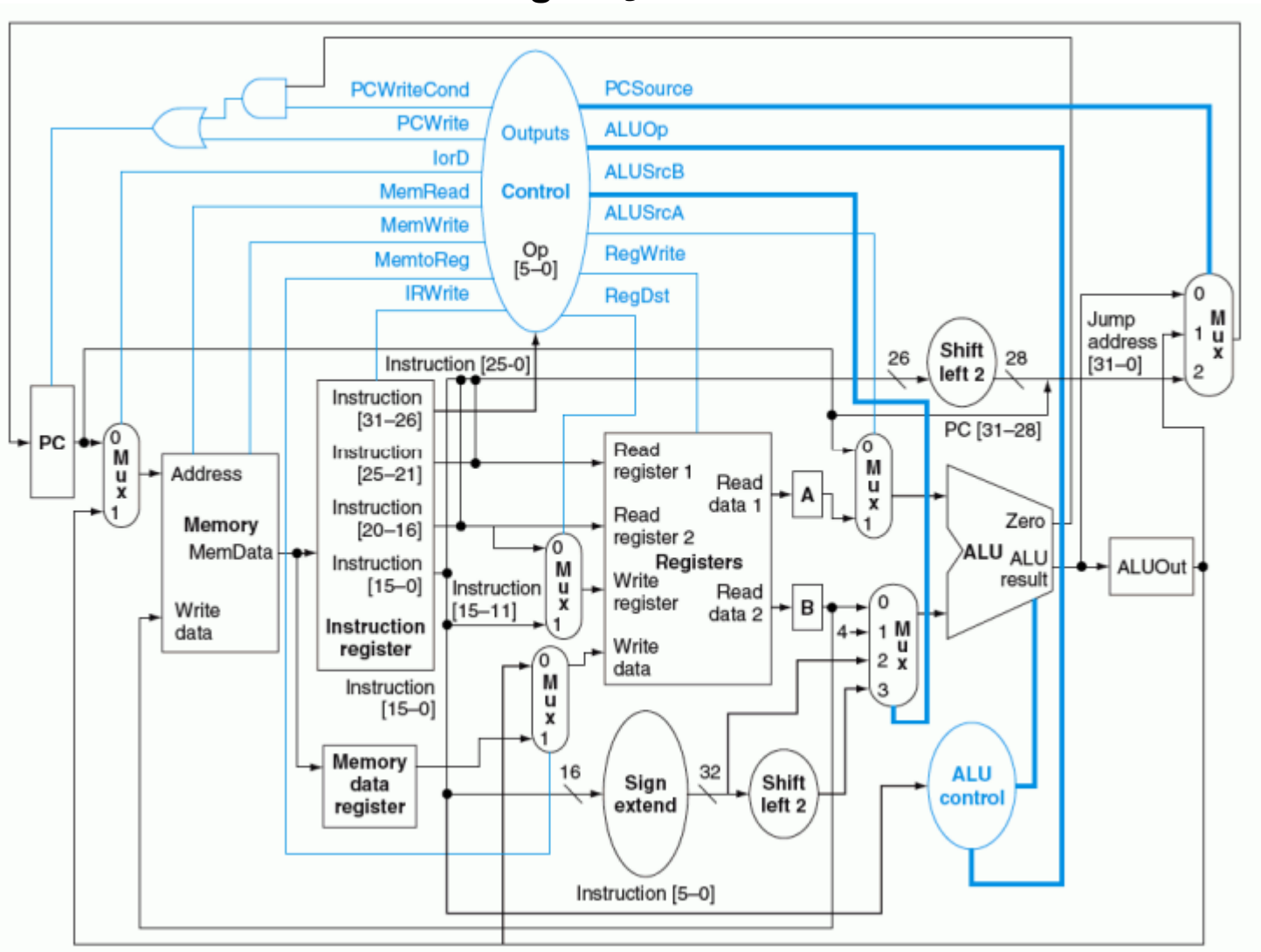
Máquina de Estados Finitos - Controle

Tipo R



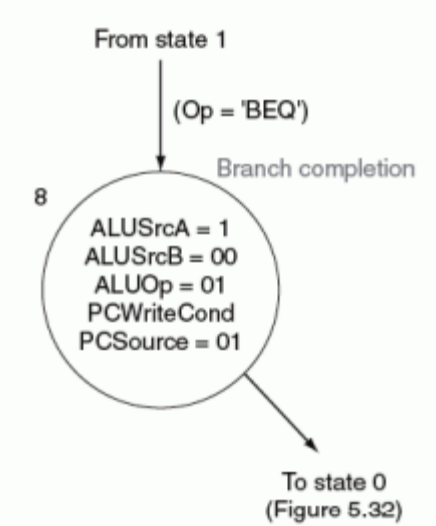
Controle Multiciclo

- **Precismos de uma nova configuração de controle**



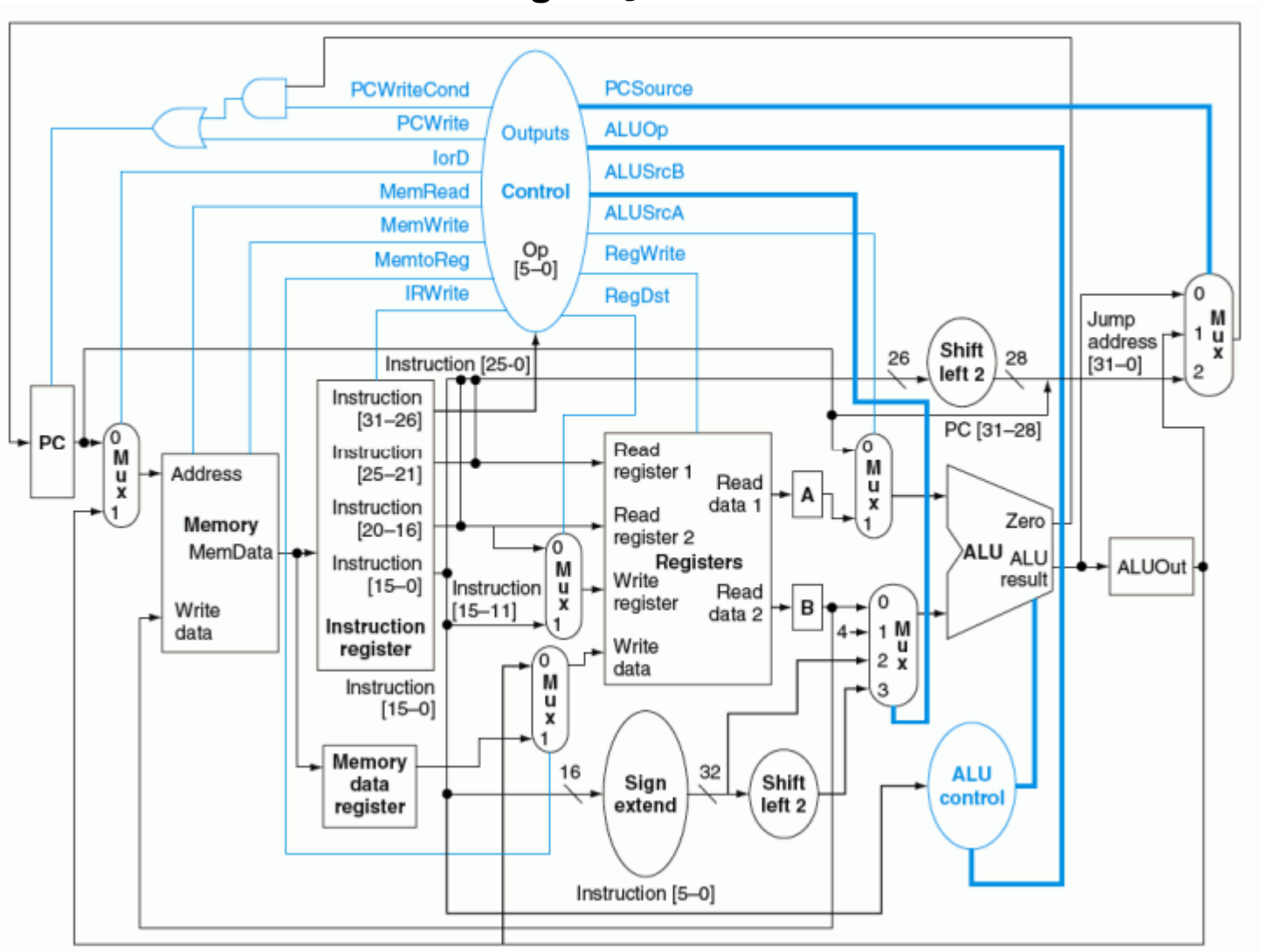
Máquina de Estados Finitos - Controle

Branch



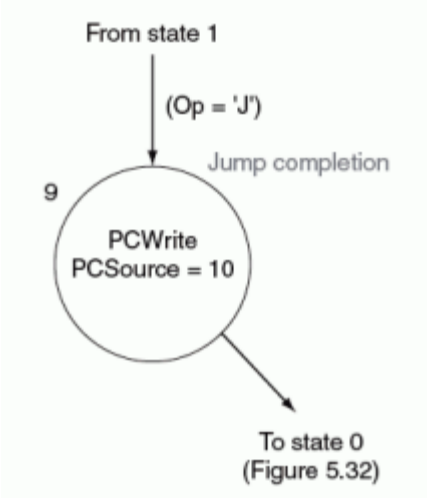
Controle Multiciclo

- **Precismos de uma nova configuração de controle**



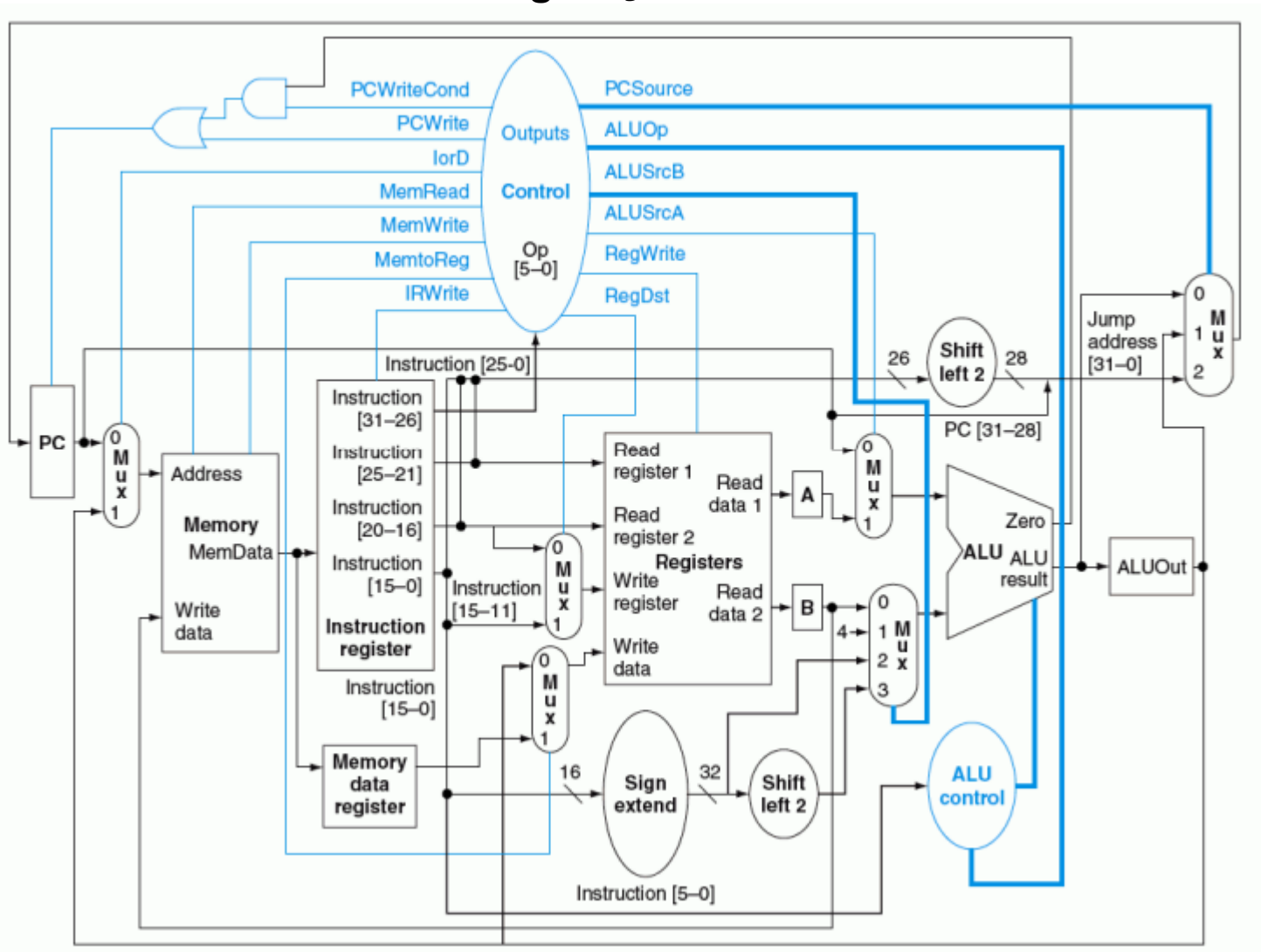
Máquina de Estados Finitos - Controle

Jump

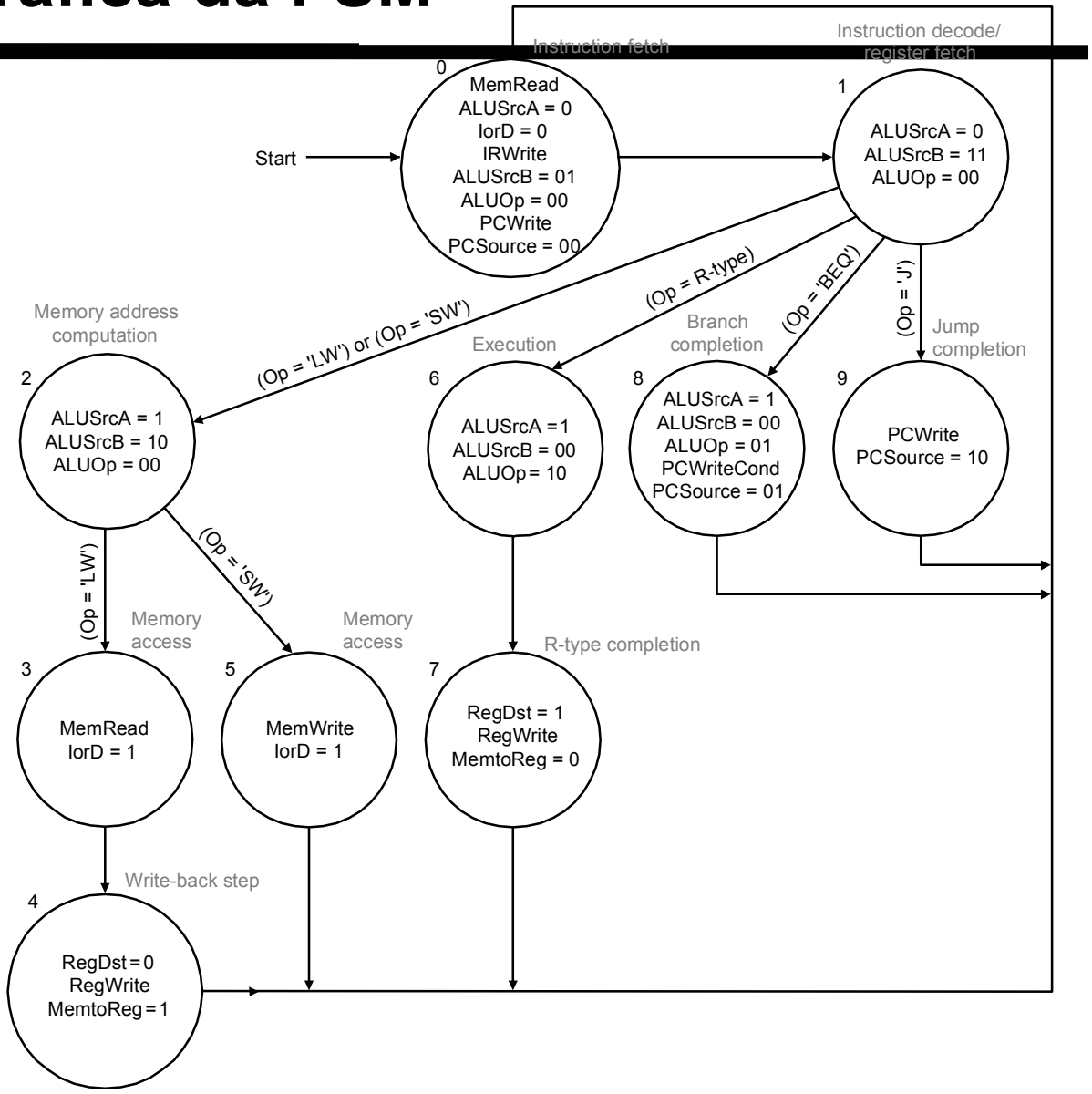


Controle Multiciclo

- **Precismos de uma nova configuração de controle**



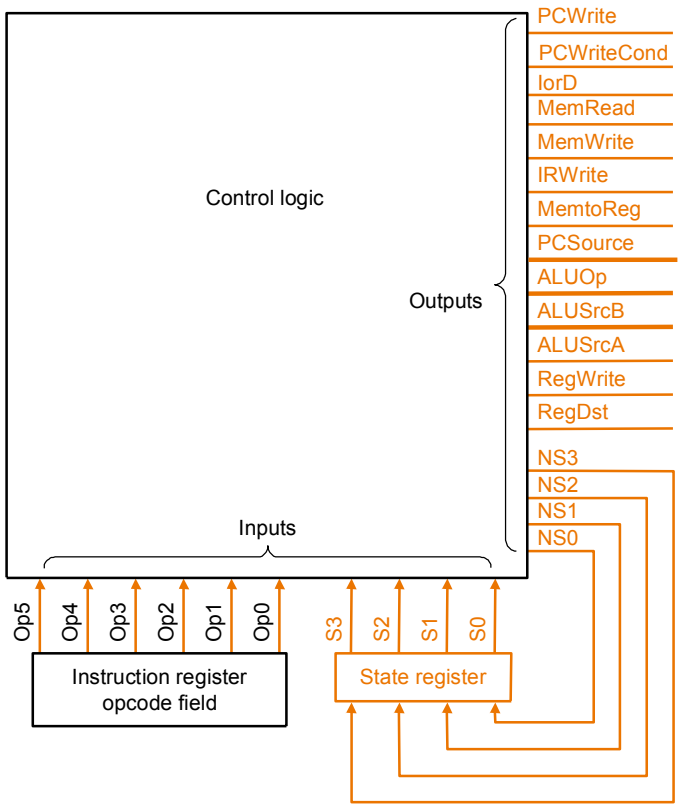
Especificação Gráfica da FSM



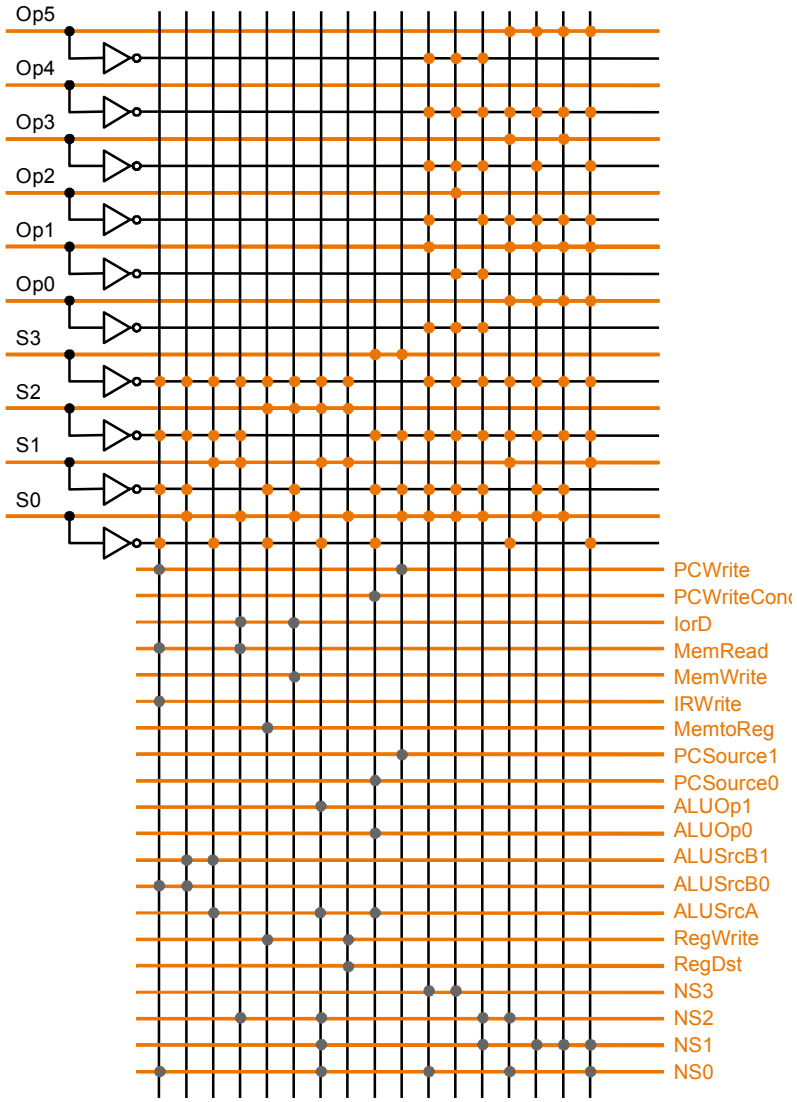
- **Quantos bits nós necessitamos para especificar os estados?**

Controle Baseado em Máquina de Estados Finitos

- **Implementação:**

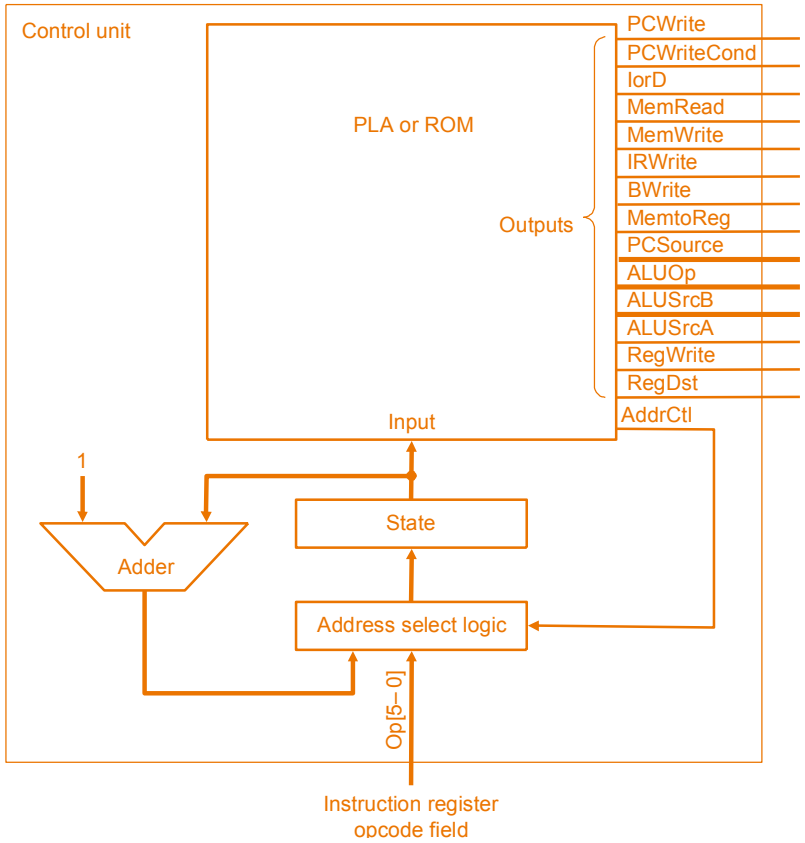


Implementação em PLA



Outro Estilo de Implementação

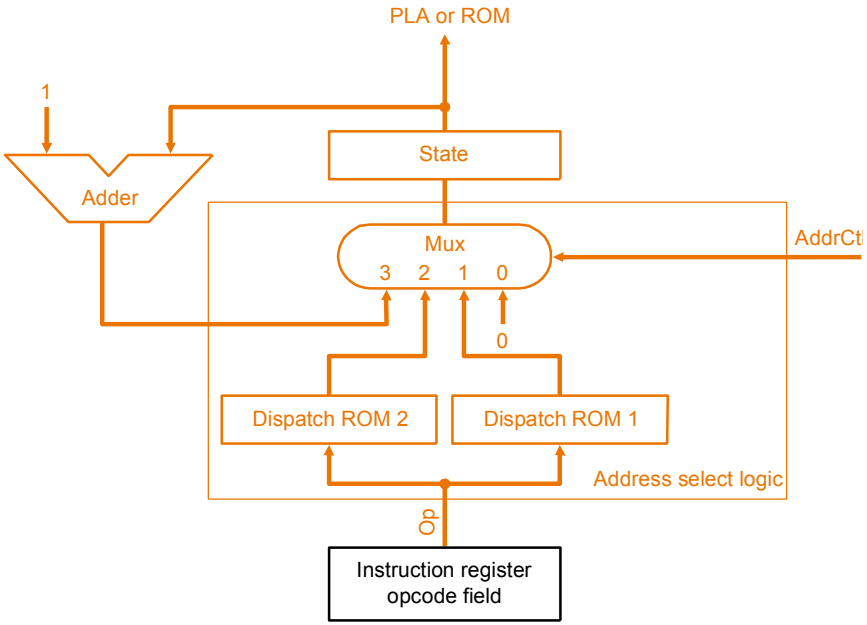
- **Instruções Complexas**: O próximo estado é frequentemente o estado corrente + 1



Detalhes

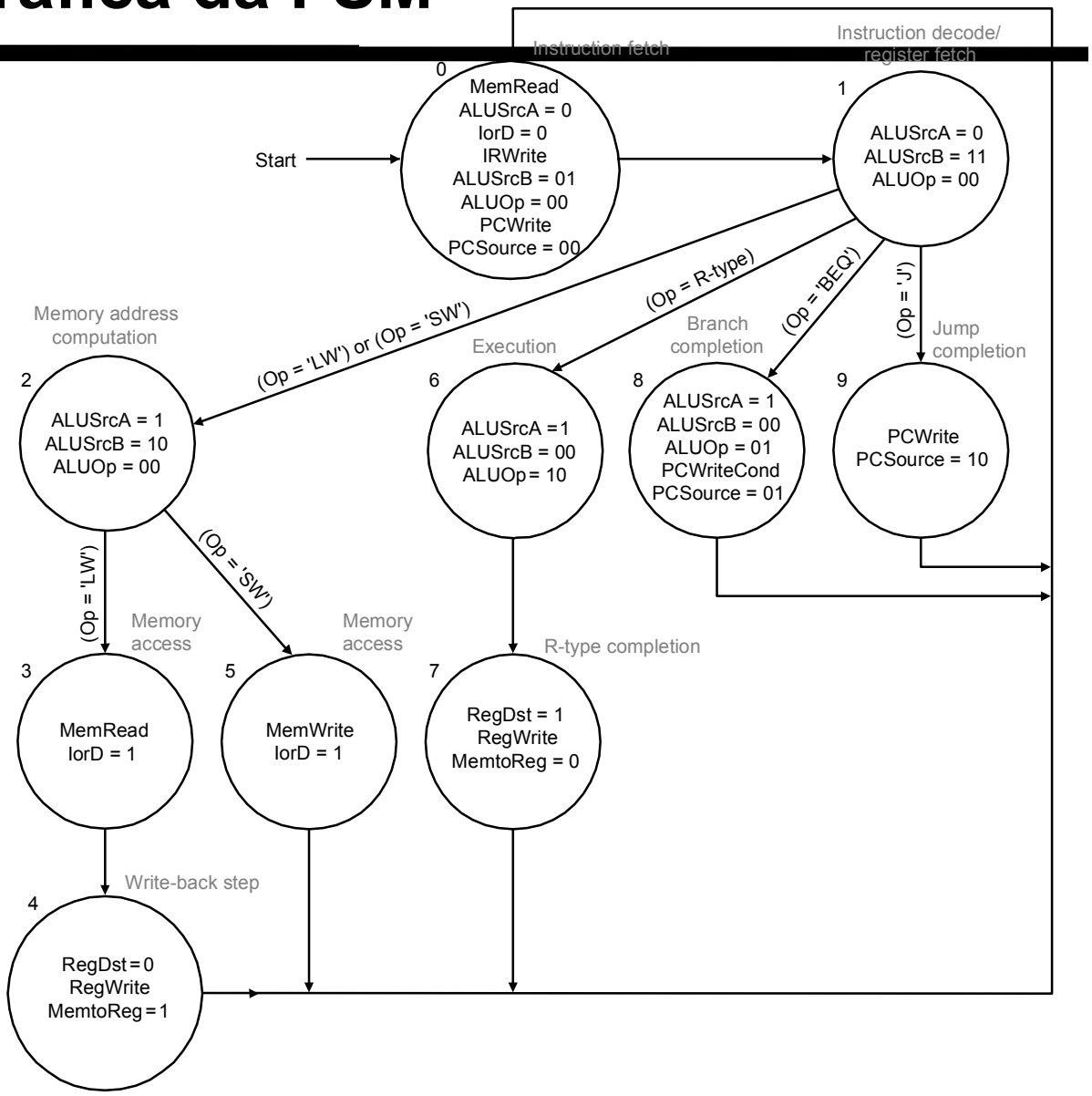
Dispatch ROM 1		
Op	Opcode name	Value
000000	R-format	0110
000010	jmp	1001
000100	beq	1000
100011	lw	0010
101011	sw	0010

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



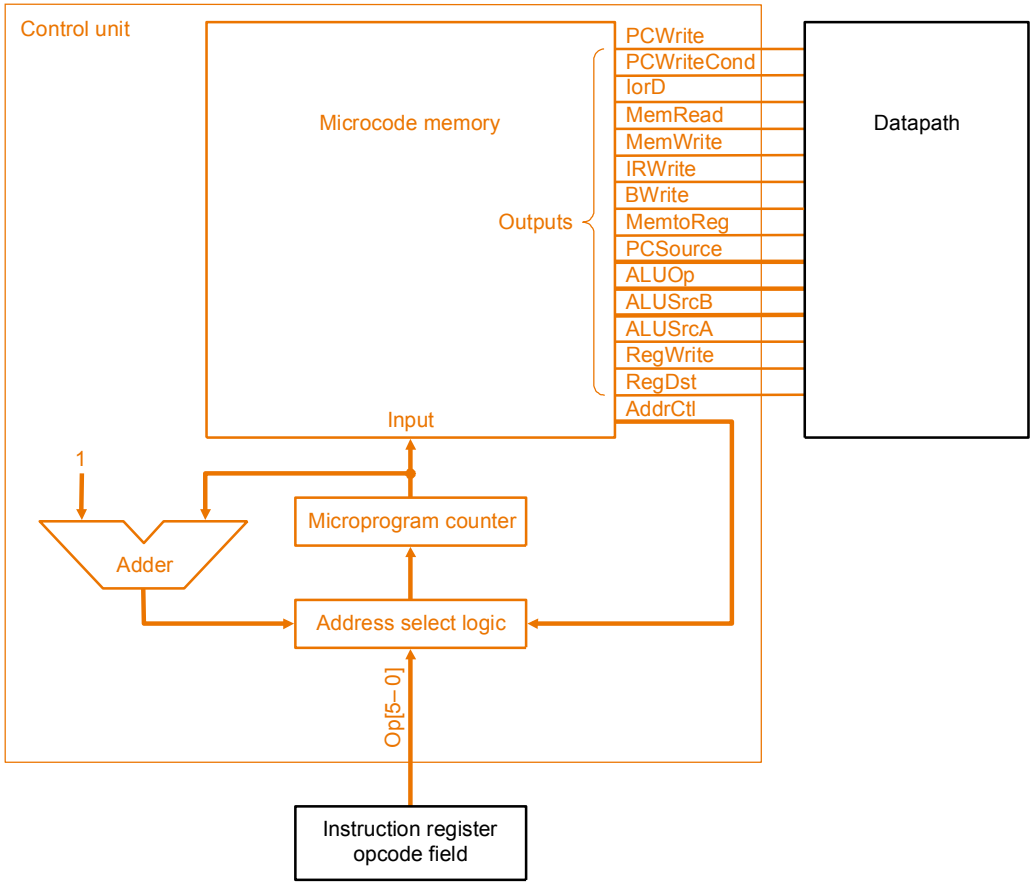
State number	Address-control action	Value of AddrCtl
0	Use incremented state	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Use incremented state	3
4	Replace state number by 0	0
5	Replace state number by 0	0
6	Use incremented state	3
7	Replace state number by 0	0
8	Replace state number by 0	0
9	Replace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

Especificação Gráfica da FSM



- **Quantos bits nós necessitamos para especificar os estados?**

Microprogramação



- **O que são "microinstruções"?**

Microprogramação

- **Campos da Microinstrução**

Nome do campo	Função do campo
ALU Control	Especificar a operação sendo realizada pela ALU durante este clock; o resultado é sempre escrito em SaídaALU.
SRC1	Especificar a origem para o primeiro operando da ALU.
SRC2	Especificar a origem para o segundo operando da ALU.
Register Control	Especificar leitura ou escrita para o banco de registradores e a origem do valor para uma escrita.
Memory	Especificar leitura ou escrita e a origem para a memória. Para uma leitura, especifica o registrador destino.
PCWrite Control	Especificar a escrita do PC.
Sequencing	Especificar como escolher a próxima microinstrução a ser executada.

Microprogramação

Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	pc	Extshft	Read			Dispatch 1

Campos	Efeito
ALU Control, SRC1, SRC2	Calcula PC + 4. (O valor também é escrito em SaídaALU, embora nunca seja lido de lá.)
Memory	Busca a instrução para IR.
PCWrite Control	Faz com que a saída da ALU seja escrita no PC.
Sequencing	Vai para a próxima microinstrução.

Campos	Efeito
ALU Control, SRC1, SRC2	Armazena PC + extensão de sinal (IR[15:0]) << 2 em ALUOut.
Register Control	Usa os campos rs e rt para ler os registradores colocando os dados em A e B.
Sequencing	Usa a tabela de despacho 1 para escolher o próximo endereço de microinstrução.

Microprogramação

Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch

Campos	Efeito
ALU Control, SRC1, SRC2	Calcula o endereço de memória: Registro (rs) + extensão de sinal (IR[15:0]), escrevendo o resultado em SaídaALU.
Sequencing	Usa a segunda tabela de despacho para desviar para a microinstrução rotulada como LW2 ou SW2.

Campos	Efeito
Memory	Lê a memória usando SaídaALU como o endereço e escreve os dados no MDR.
Sequencing	Vai para a próxima microinstrução.

Campos	Efeito
Register Control	Escreve o conteúdo do MDR na entrada do banco de registradores especificada por rt.
Sequencing	Vai para a microinstrução rotulada como Fetch.

Campos	Efeito
Memory	Escreve na memória usando o conteúdo de SaídaALU como o endereço e o conteúdo de B como o valor.
Sequencing	Vai para a microinstrução rotulada como Fetch.

Microprogramação

Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch

Campos	Efeito
ALU Control, SRC1, SRC2	A ALU opera com os conteúdos dos registradores A e B, usando o campo Function para especificar a operação da ALU.
Sequencing	Vai para a próxima microinstrução.

Campos	Efeito
Register Control	O valor em SaídaALU é escrito na entrada do banco de registradores especificada pelo campo rd.
Sequencing	Vai para a microinstrução rotulada como Fetch.

Microprogramação

Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
BEQ1	Subt	A	B			ALUOut-cond	Fetch

Campos	Efeito
ALU Control, SRC1, SRC2	A ALU subtrai os operandos em A e B para gerar a saída Zero.
PCWrite Control	Faz com que o PC seja escrito usando o valor já em SaídaALU, se a saída Zero da ALU for verdadeira.
Sequencing	Vai para a microinstrução rotulada como Fetch.

Microprogramação

Label	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
JUMP1						Jump Address	Fetch

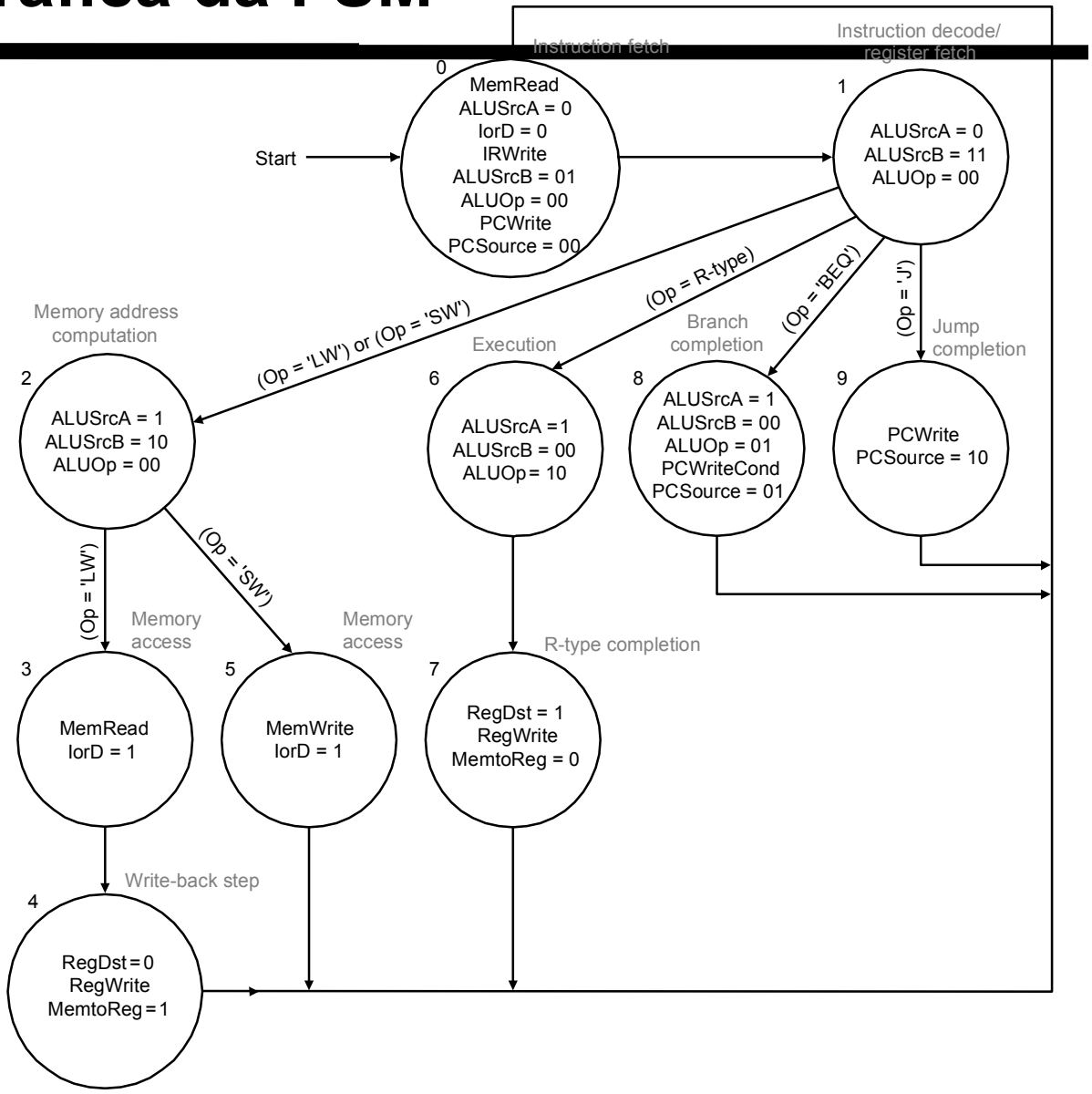
Campos	Efeito
PCWrite Control	Faz com que o PC seja escrito usando o endereço de destino de desvio.
Sequencing	Vai para a microinstrução rotulada como Fetch.

Microprogramação

- Uma metodologia de especificação
  - apropriada para 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			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

Especificação Gráfica da FSM



- **Quantos bits nós necessitamos para especificar os estados?**

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 Formato Máximo e Mínimo

- **Sem codificação:**
  - **1 bit para cada operação no datapath**
  - **rápido, requer mais memória (lógica)**
  - **usado para Vax 780 — 400K de memória!**
- **Codificação:**
  - **Sinais de controle são gerados pela microinstrução e lógica**
  - **usa menos memória, mais lento**
- **Contexto histórico do CISC:**
  - **Muita lógica para colocar em um único chip**
  - **Usa uma ROM (ou mesmo RAM) para o microcódigo**
  - **É fácil adicionar novas instruções**

Microcódigo: Compromissos

- Distinção entre especificação e implementação é por vezes nebulosa

- Vantagens da especificação:
  - Facilidade para projetar e escrever
  - Projeto da arquitetura e microcódigo em paralelo
- Implementação (ROM) vantagens
  - facilidade para mudar os valores que estão na memória
  - Pode emular outras arquiteturas
  - Pode fazer uso de registradores internos
- Desvantagens, LENTO pois:
  - O controle é implementado no mesmo chip do processador

Figura Descritiva