CAPÍTULO 4
CAMINHO DE DADOS E CONTROLE

- Introdução
- Uma implementação MIPS básica
- Sinopse da implementação
- Sinais de controle
- Multiplexadores (muxes)
- Implementação monociclo
- Metodologia de clocking
- Construindo um caminho de dados monociclo
- Unidade de Controle monociclo
- Operação do caminho de dados
- Implementação Multiciclo
- CPI em uma CPU multiciclo
- Unidade de Controle multiciclo
- Exceções

Introdução

- Vimos que o desempenho depende de:
  - Contagem de instruções
  - Ciclo de clock
  - CPI

  } Afetados pela implementação do conj. de instruções

- Neste capítulo vamos ver como é a implementação do conjunto de instruções
- Duas implementações:
  - Monociclo
  - Multiciclo

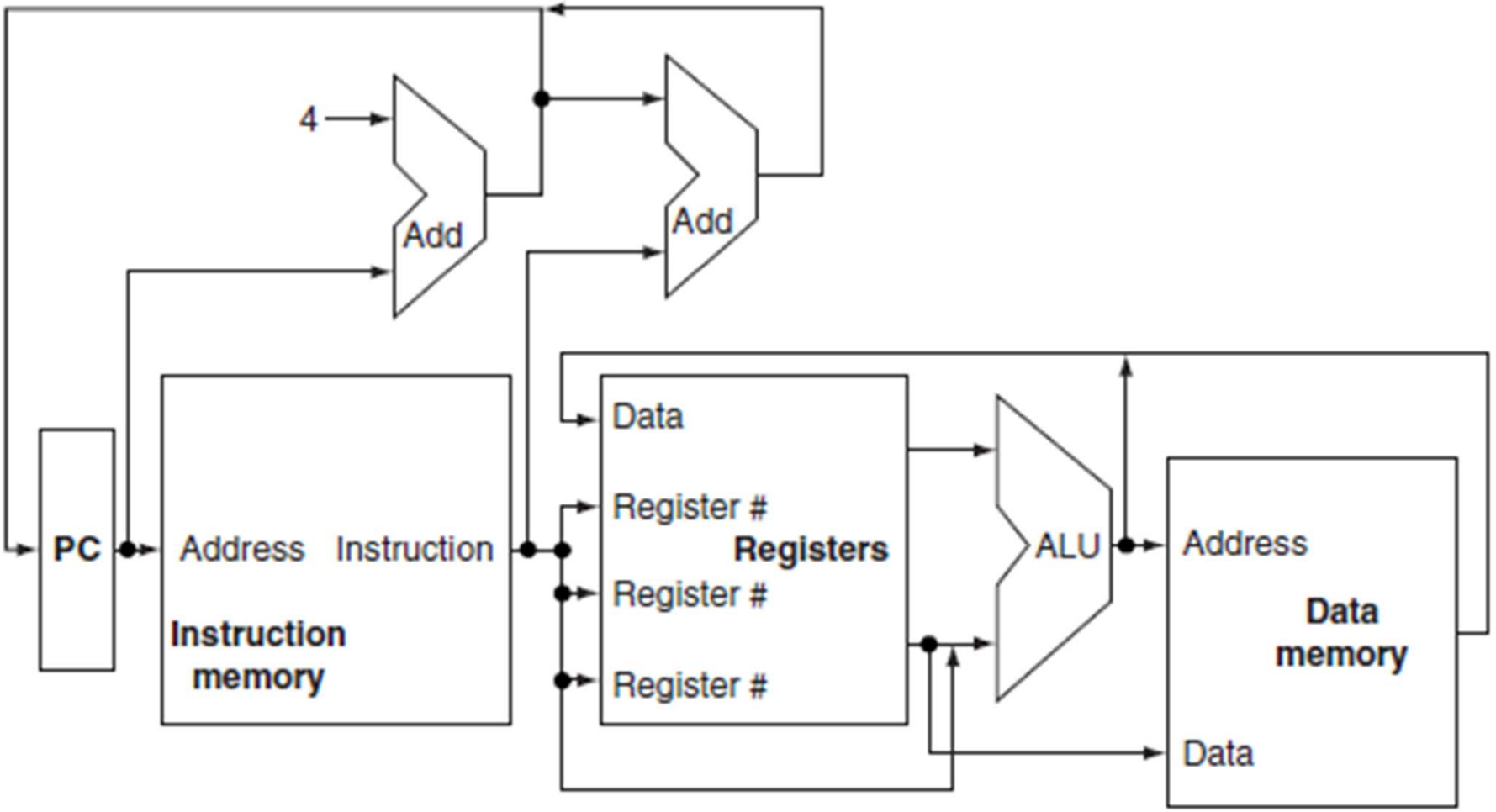
Uma implementação MIPS básica

- Uma implementação MIPS básica considera:
  - Instruções load word (lw) e store word (sw)
  - Aritméticas: add, sub, and e slt
  - Desvio: branch if equal (beq) e jump (j)
- Não inclui muitas instruções (ex.: shift, mult, div, op. ponto flutuante, etc.), mas demonstra exatamente como é criado um caminho de dados (datapath) e controle.
- Entenderemos como a implementação afeta o ciclo de clock e a CPI do processador.

Sinopse da implementação

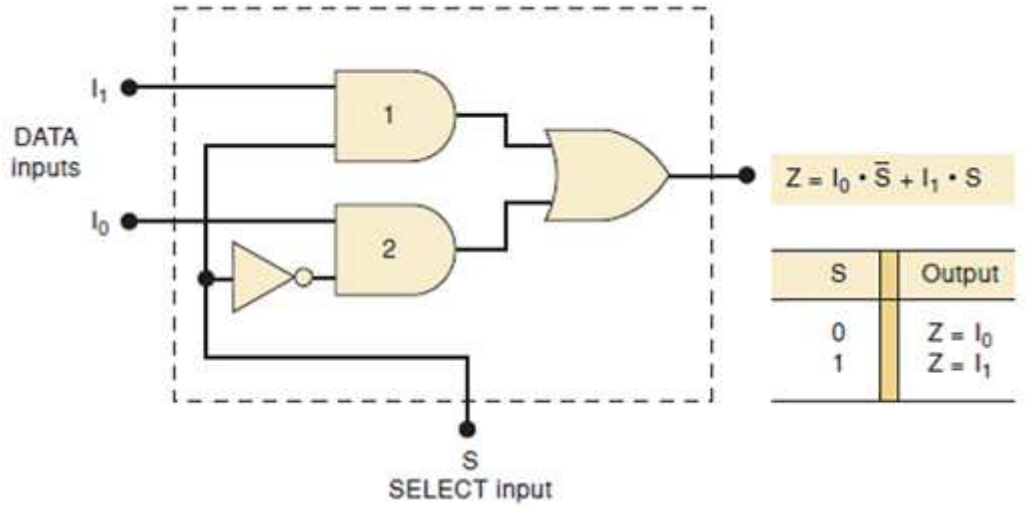
- Cada tipo de instrução realiza um conjunto de ações no caminho de dados
- Porém, as duas primeiras etapas são idênticas para todas as instruções:
  - Buscar a instrução na memória na posição indicada pelo contador de programa (PC)
  - Ler os registradores indicados na instrução (um ou dois)
- Após isso, muda ligeiramente para cada instrução
- Mas há semelhanças: quase todas usam a ALU
- As unidades funcionais se interconectam de maneira a contemplar todas as classes de instrução

Sinopse da implementação



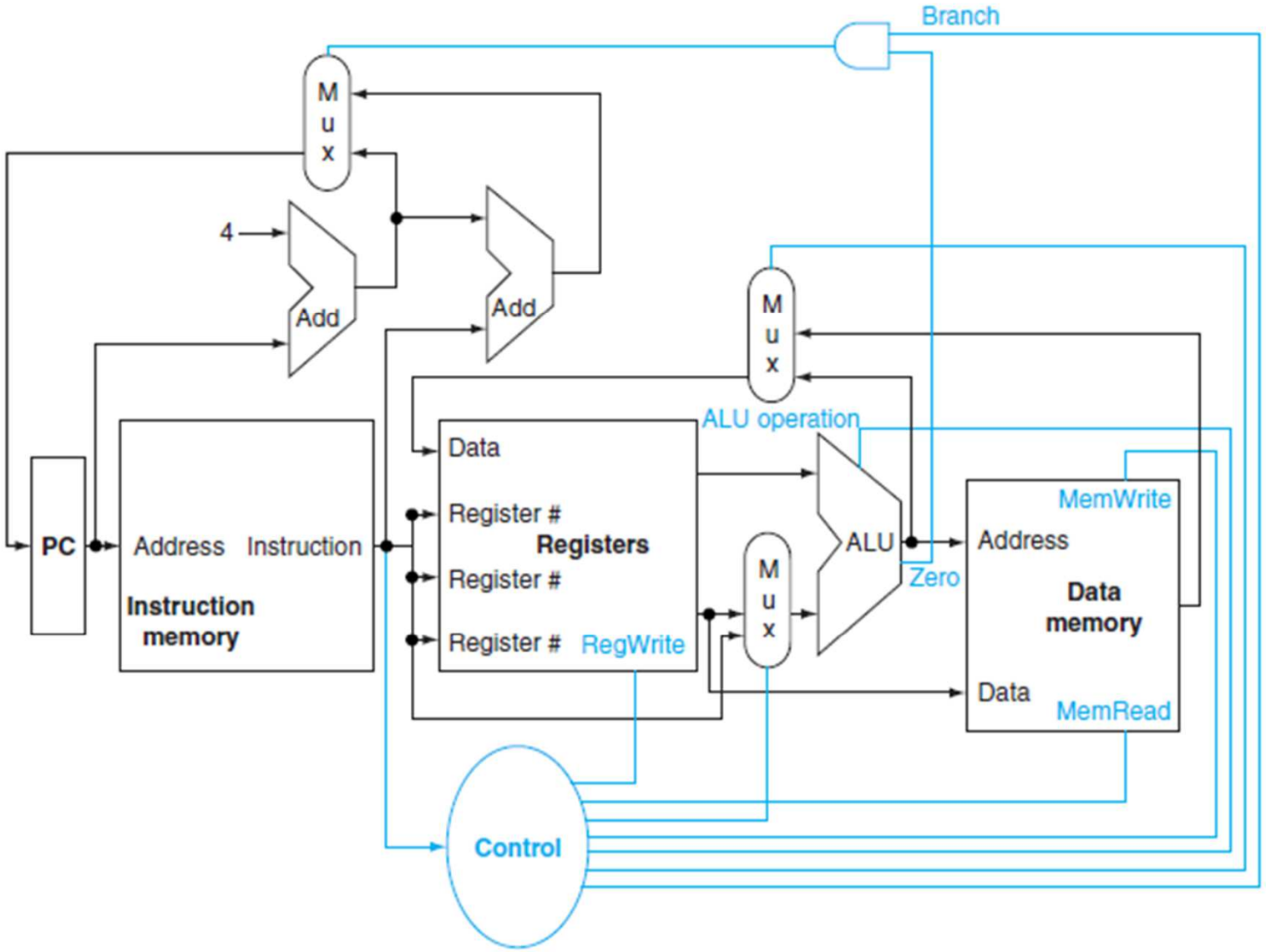
Multiplexadores (muxes)

- São circuitos digitais seletores de dados
- São usados para desviar os dados de acordo com sinais de controle
- Os sinais de controle são gerados de acordo com as instruções para controle das unidades funcionais e dos muxes.
- Multiplexador básico de duas entradas:



- 32 desses podem ser usados juntos para fazer um mux de 32 bits

Sinais de controle



Implementação monociclo

- A arquitetura que estamos definindo é monociclo. Embora seja mais simples, não é utilizado, pois possui desvantagens:
  - É mais lento que a arquitetura multiciclo (ciclo maior)
  - Precisa duplicar unidades funcionais que forem necessárias usar simultaneamente
  - Não permite fazer pipeline

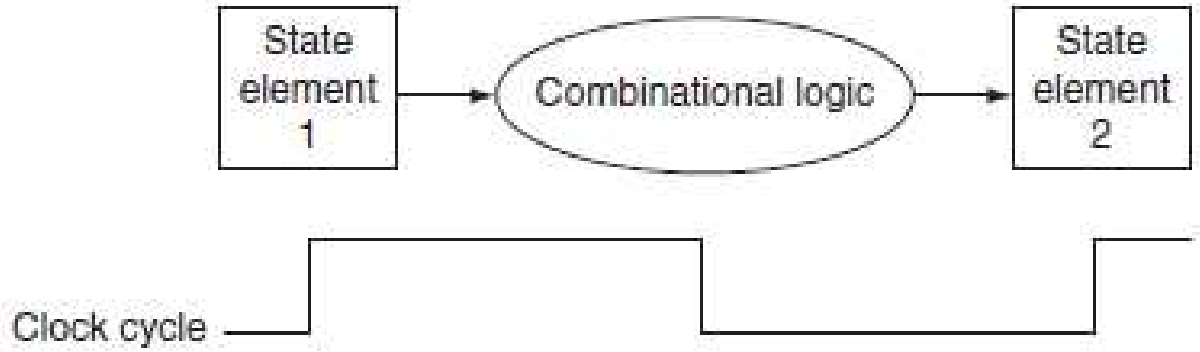
- A implementação usa lógica combinacional (circuitos que operam nos valores dos dados. Ex: ALU) e sequenciais
- Os sequenciais também são chamados elementos de estado. São os que possuem memória.

Metodologia de clocking

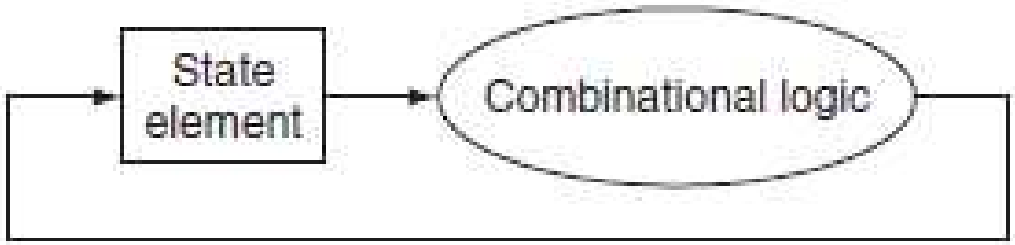
- Define o momento de escrita e leitura de dados.
- É a abordagem usada para determinar quando os dados são válidos e estáveis em relação ao clock.
- A saída de um elemento de estado fornece o valor escrito em um ciclo anterior.
- A leitura e escrita não podem ocorrer ao mesmo tempo, pois um dado antigo poderia ser lido.
- Os eventos ocorrem nas transições do clock, positiva ou negativa.
- A lógica combinatória, os elementos de estado e o clock estão intimamente relacionados.

Metodologia de clocking

- A duração do ciclo de clock depende da estabilização do resultado da lógica combinacional

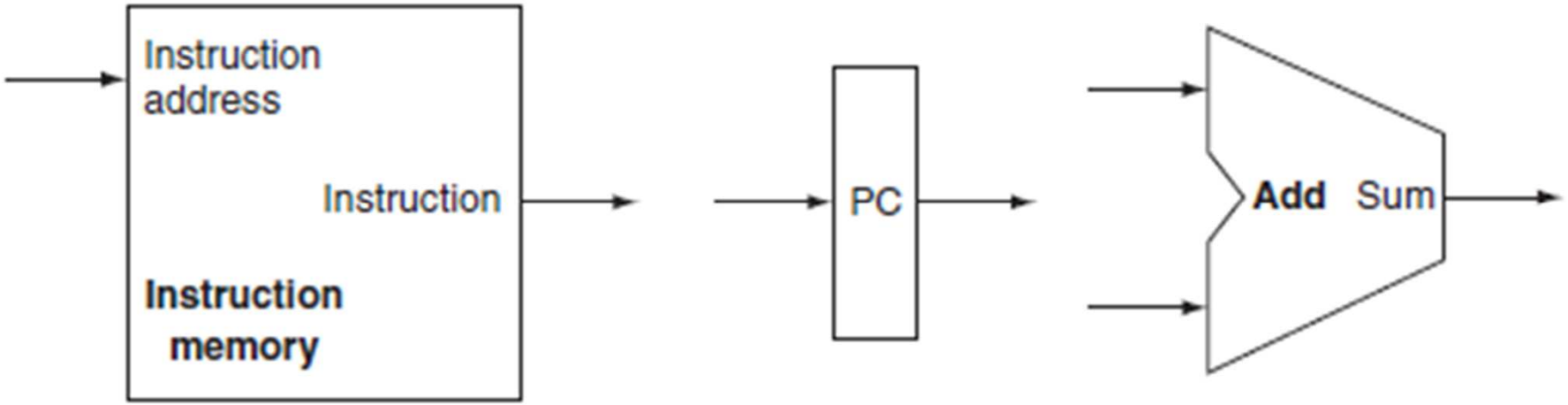


- O acionamento por transição permite que um elemento de estado seja lido e escrito no mesmo ciclo de clock.



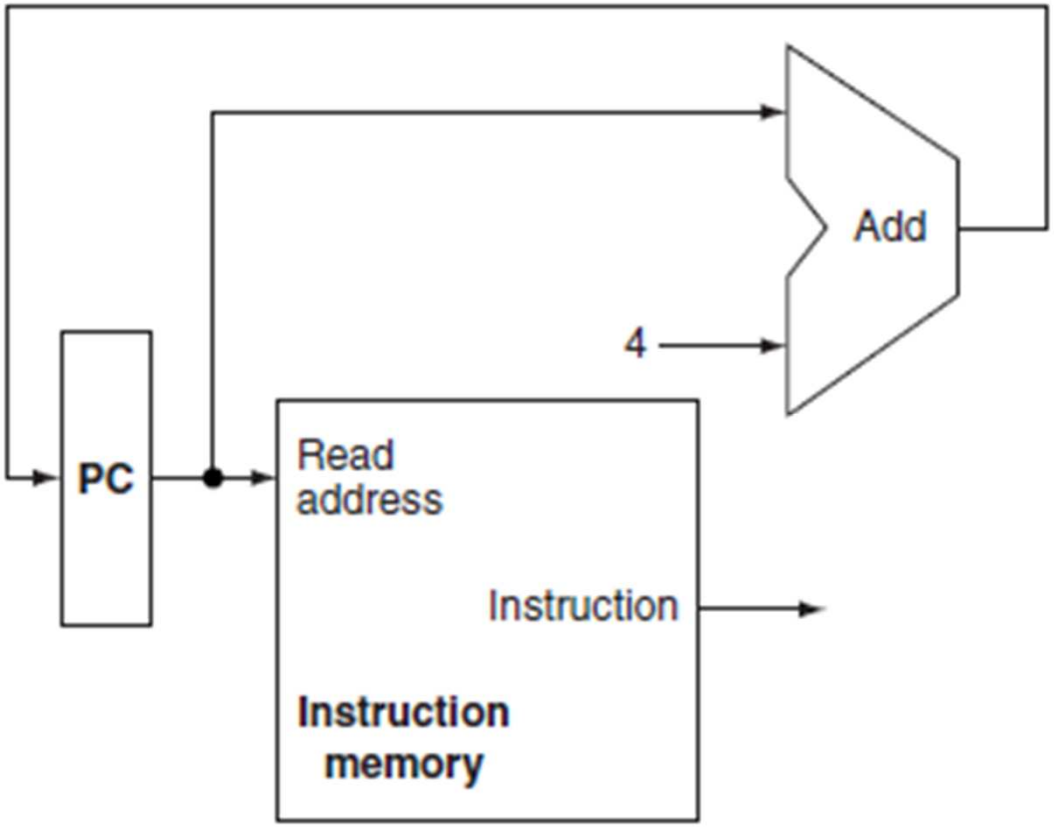
Construindo um caminho de dados

- Vamos analisar quais elementos do caminho de dados cada instrução precisa para sua execução
- Três elementos iniciais executam os dois passos citados que são iguais para toda instrução:
  - Memória de instruções
  - Contador de programa (PC)
  - Somador



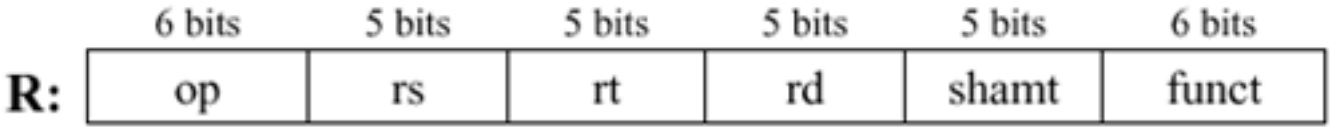
Construindo um caminho de dados

- A parte inicial do caminho de dados usada para buscar a instrução (Instruction Fetch) utiliza o somador para determinar o endereço da próxima instrução:



Construindo um caminho de dados

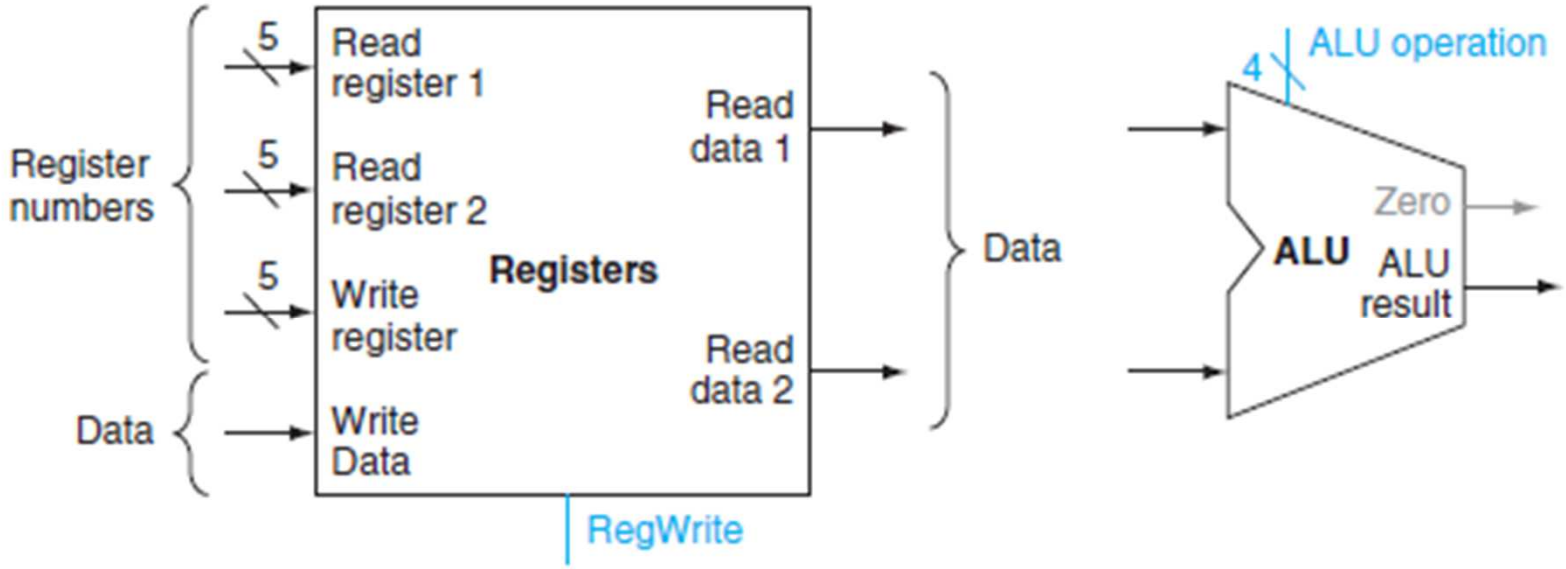
- Vamos considerar uma instrução do tipo R.
- Relembrando:



- Um caso típico é **add $t1, $t2, $t3**. São portanto necessários dois registradores fonte. O banco de registradores deve permitir a leitura de dois registradores simultaneamente.
- Como o campo de cada registrador é de 5 bits, o banco pode possuir até $2^5 = 32$ registradores

Construindo um caminho de dados

- Os elementos necessários para implementar as instruções tipo R são:
  - Banco de registradores
  - ALU

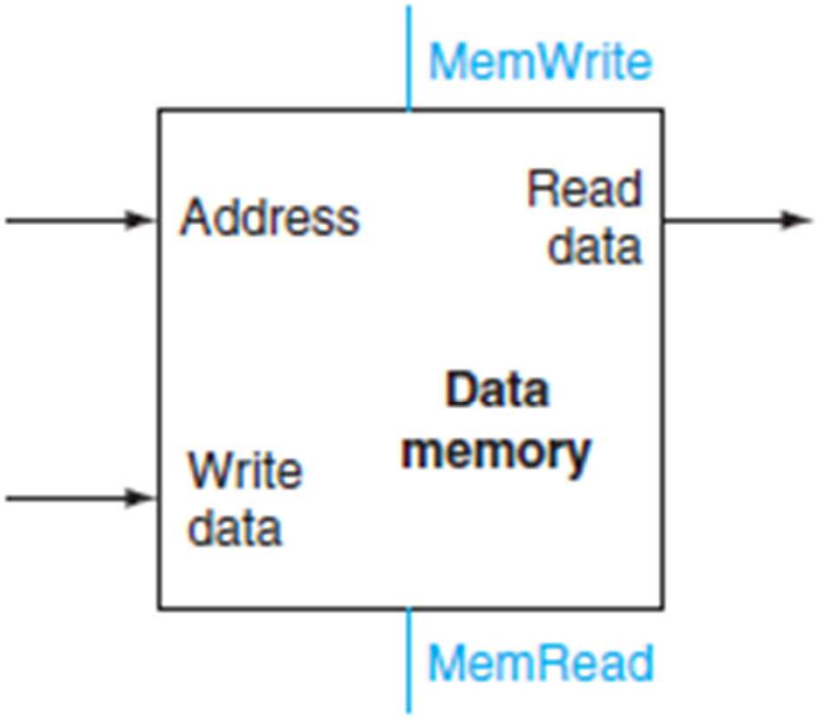


Construindo um caminho de dados

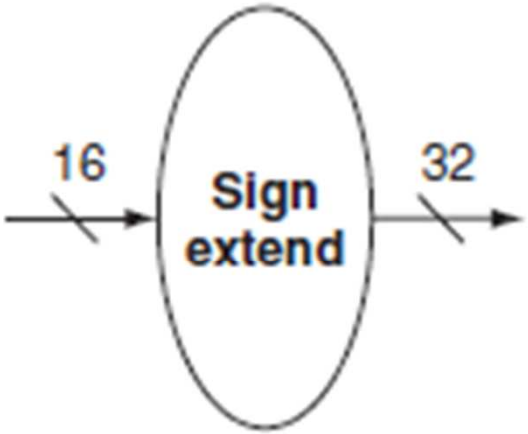
- Agora consideremos as instruções load/store com formato:
- `lw $t1,offset_value($t2)` ou `sw $t1,offset_value($t2)`.
- O valor sinalizado de 16 bits do offset_value deve ser somado com o valor de um registrador para resultar no endereço do operando
- Assim, load e store também usam a ALU
- Além disso, precisamos de uma unidade para estender o sinal do campo de 16 bits

Construindo um caminho de dados

- Elementos necessários para as instruções load/store:



a. Data memory unit



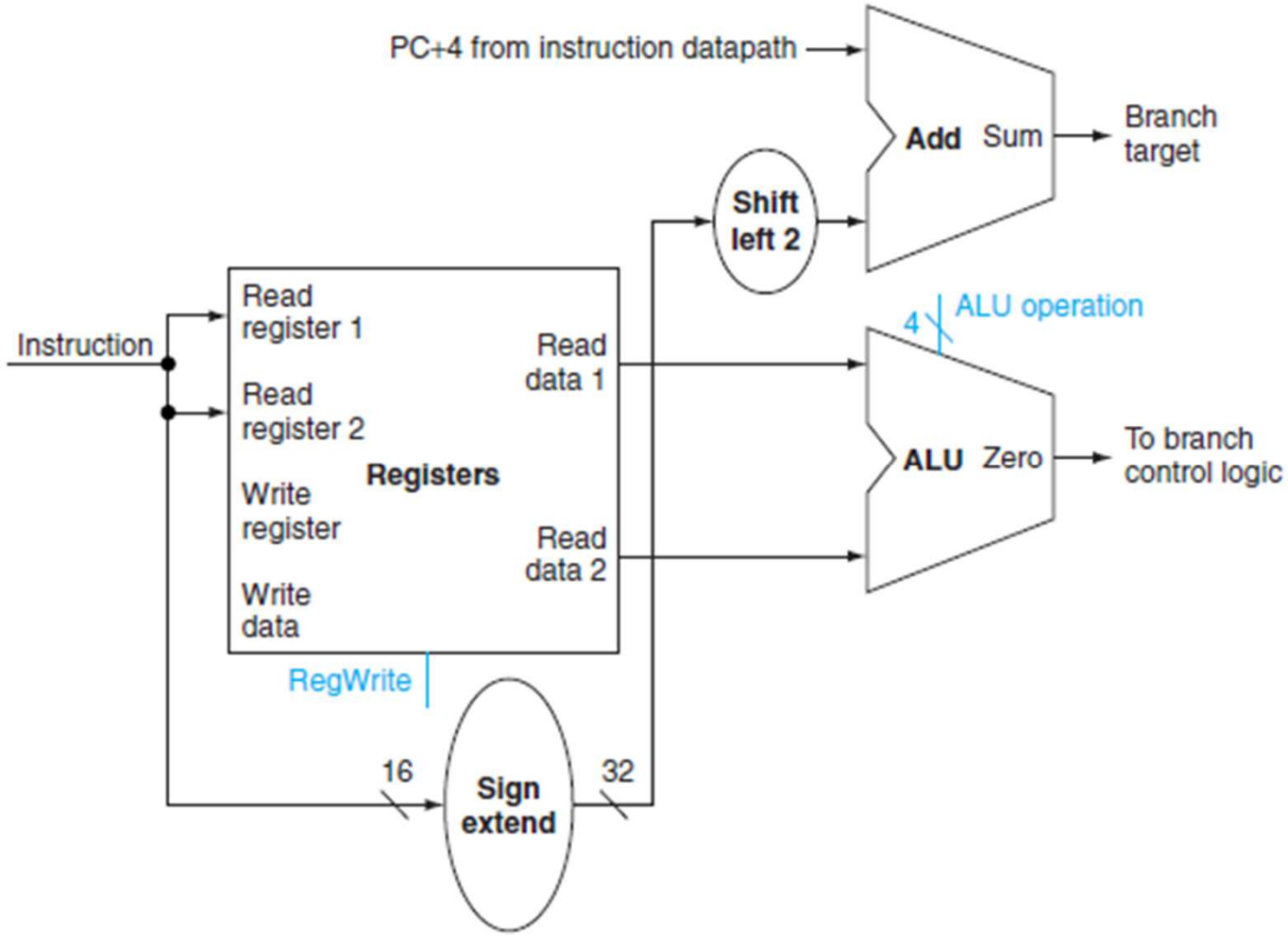
b. Sign-extension unit

Construindo um caminho de dados

- A instrução beq é semelhante, pois também possui um offset que precisa estender o bit de sinal, porém o registrador origem é o PC.
- Porém tem alguns detalhes extras:
  - Deve-se usar PC+4 como base para o desvio
  - O campo offset deve ser deslocado de 2 bits
- Os registradores especificados devem ser levados até a ALU para realizar a comparação e um sinal (bit zero) vai comandar um mux para decidir por **desvio tomado** e **desvio não tomado**.

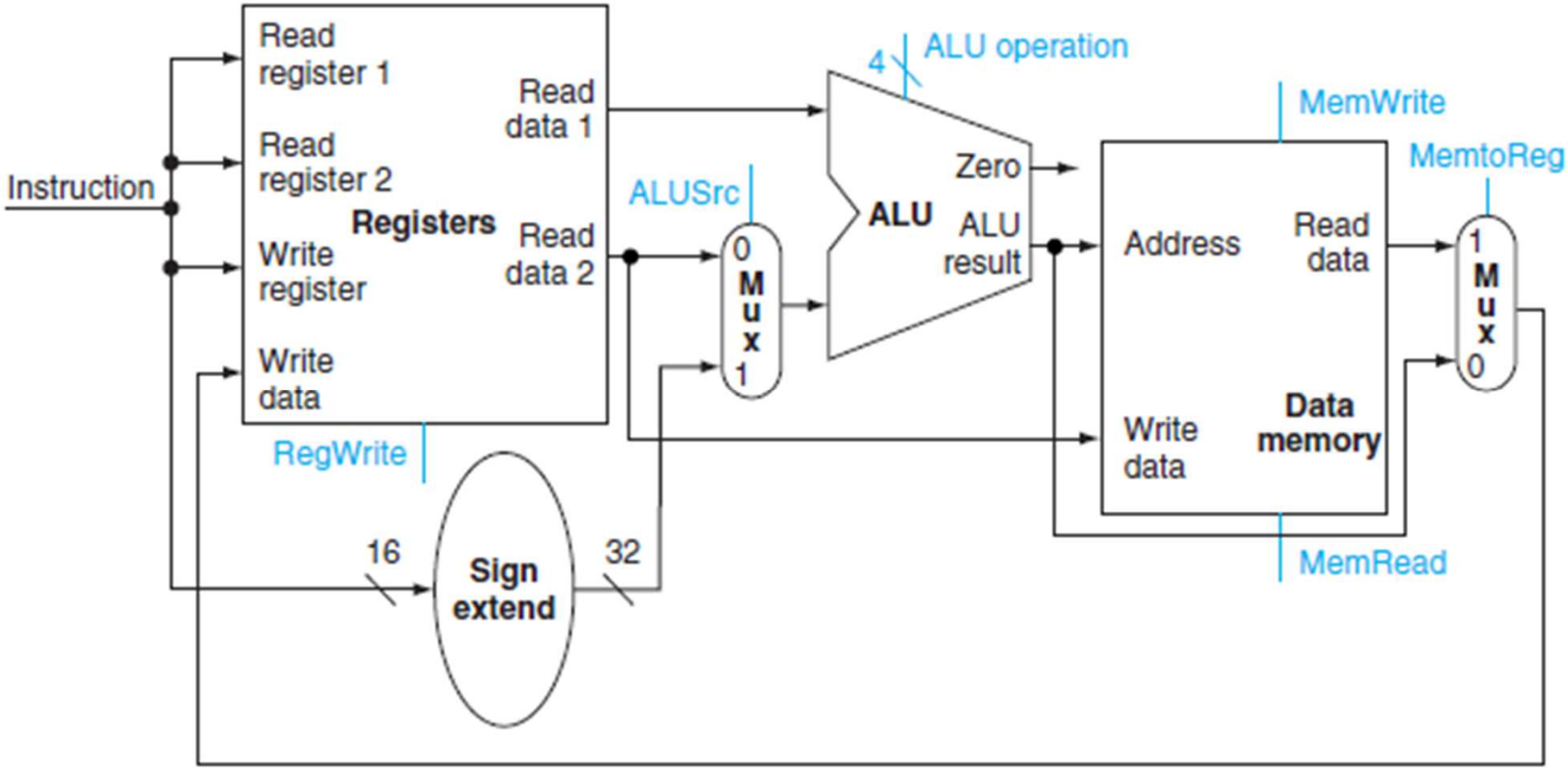
Construindo um caminho de dados

- O caminho de dados para um desvio condicional (beq)



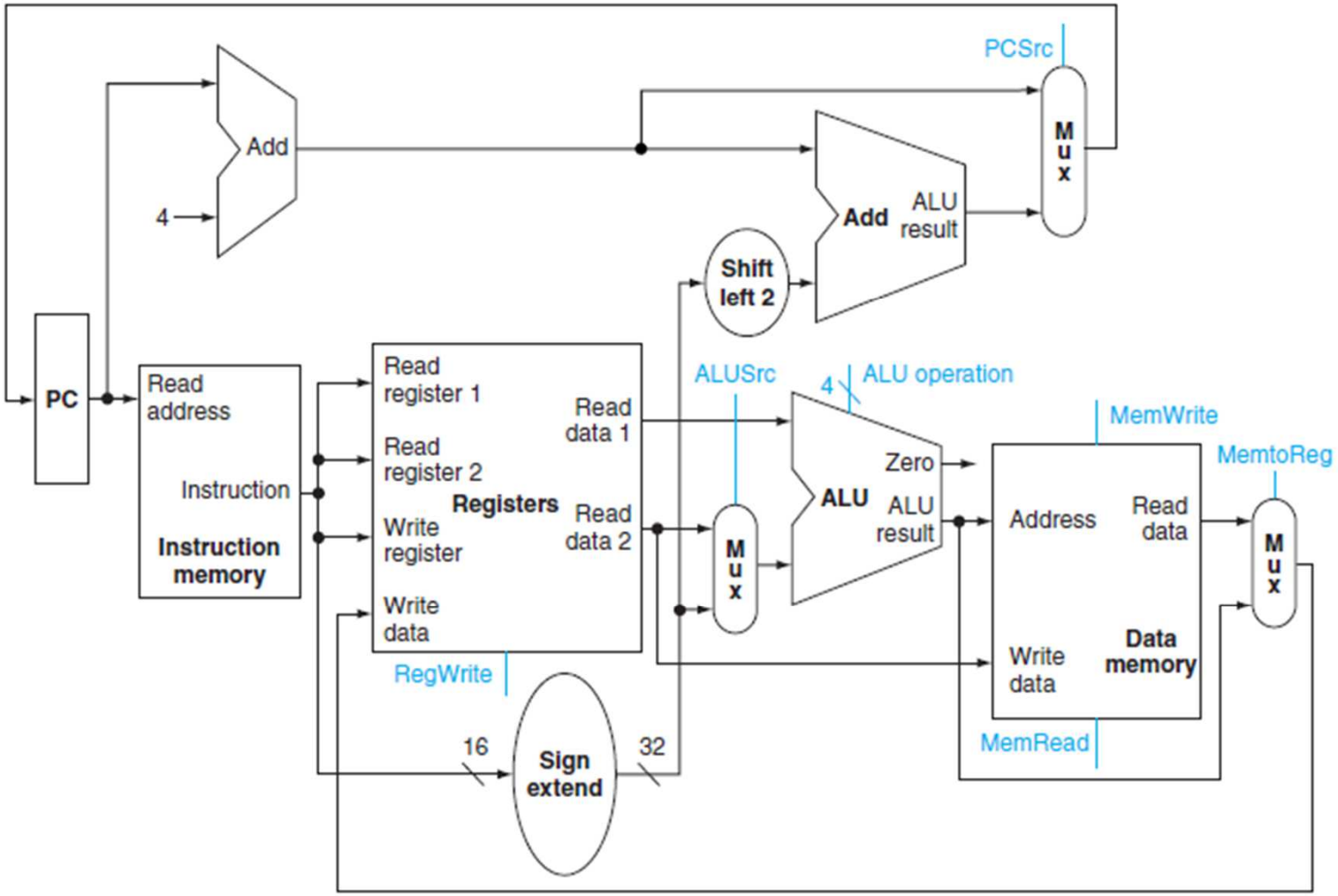
Construindo um caminho de dados

- O caminho de dados combinado para instruções de acesso à memória e instruções tipo R:



Construindo um caminho de dados

- Caminho de dados:



Unidade de Controle

- A ALU proposta tem 4 entradas de controle e realiza 6 operações:

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

- A ALU pode ter uma unidade de controle própria que recebe como entrada um campo de 2 bits da unidade de controle principal (OpALU) e o campo funct da instrução tipo R

Unidade de Controle

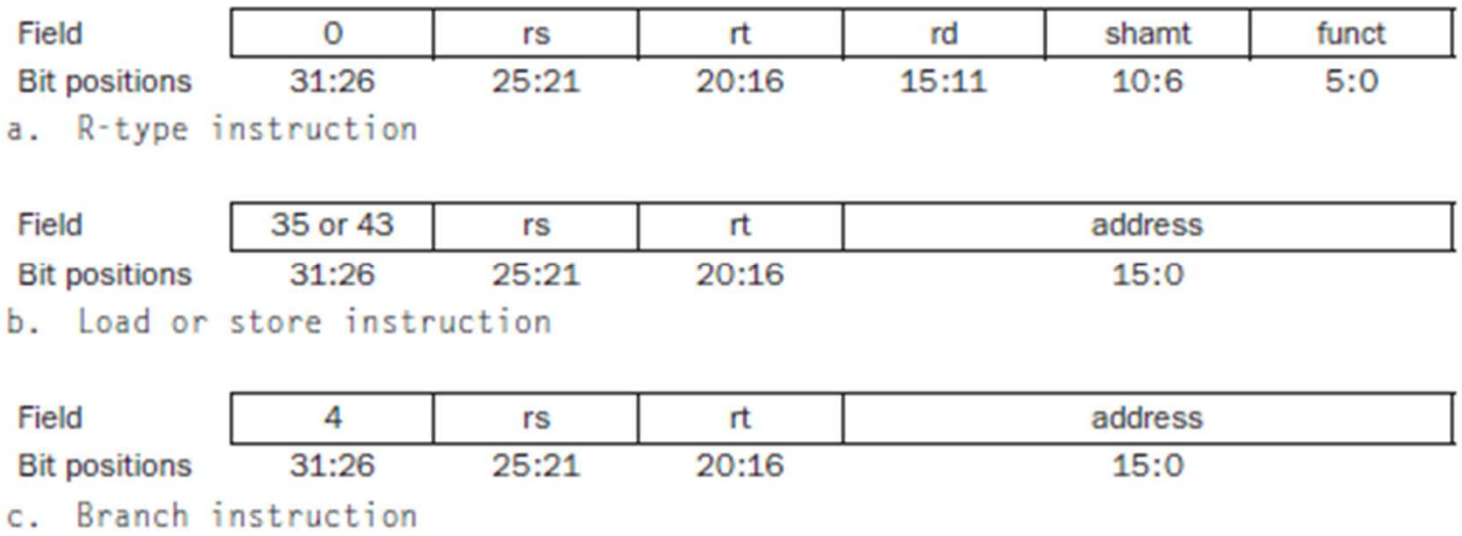
- Geração dos bits de controle da ALU:

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	and	0000
R-type	10	OR	100101	or	0001
R-type	10	set on less than	101010	set on less than	0111

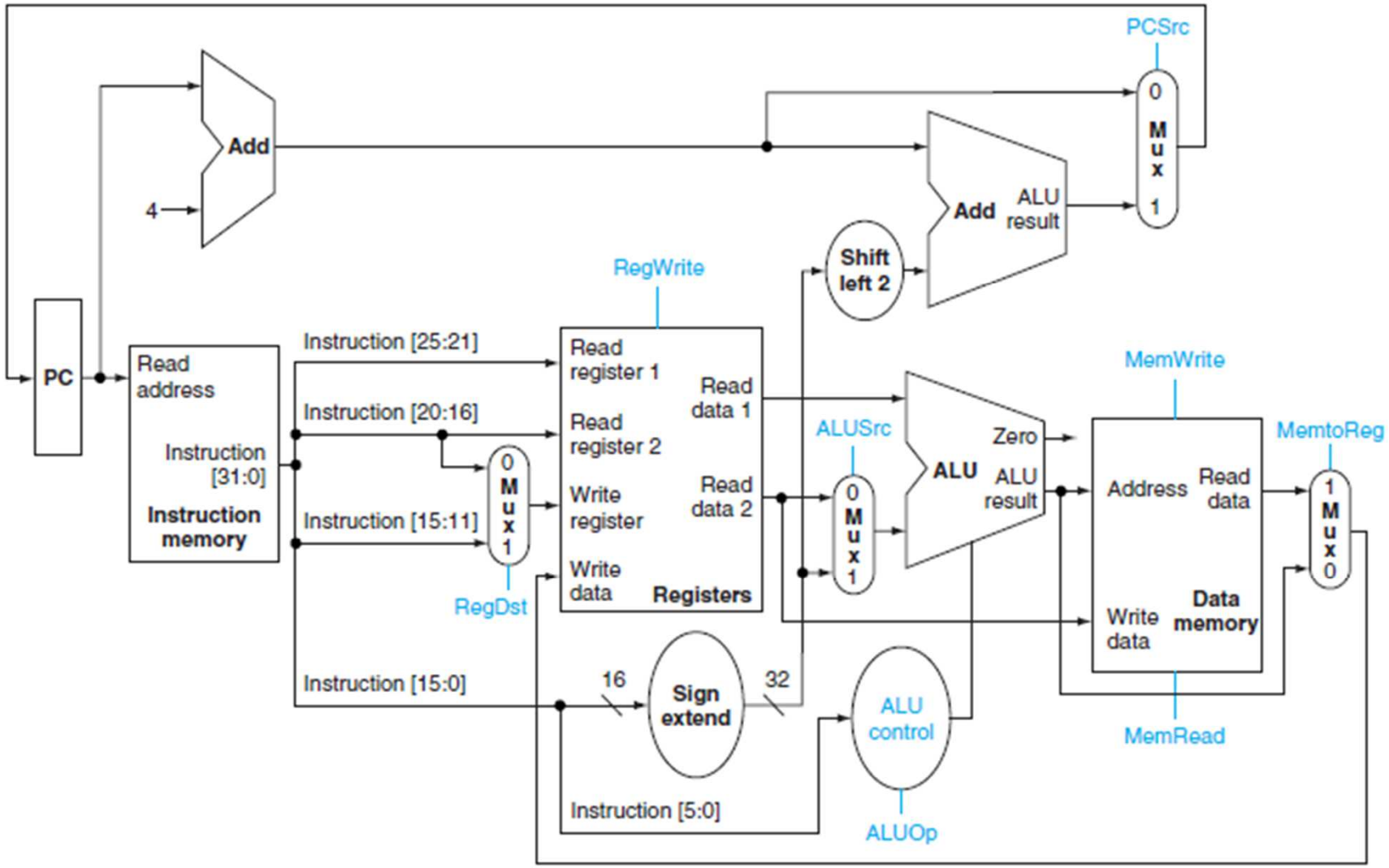
ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

Unidade de Controle

- Para projetar a unidade de controle, é necessário identificar os campos das instruções e as linhas de controle necessárias
- O registrador destino pode vir da faixa 15:11 ou 20:16, dependendo da instrução.
- Portanto é preciso adicionar mais um mux no datapath anterior



Unidade de Controle



Unidade de Controle

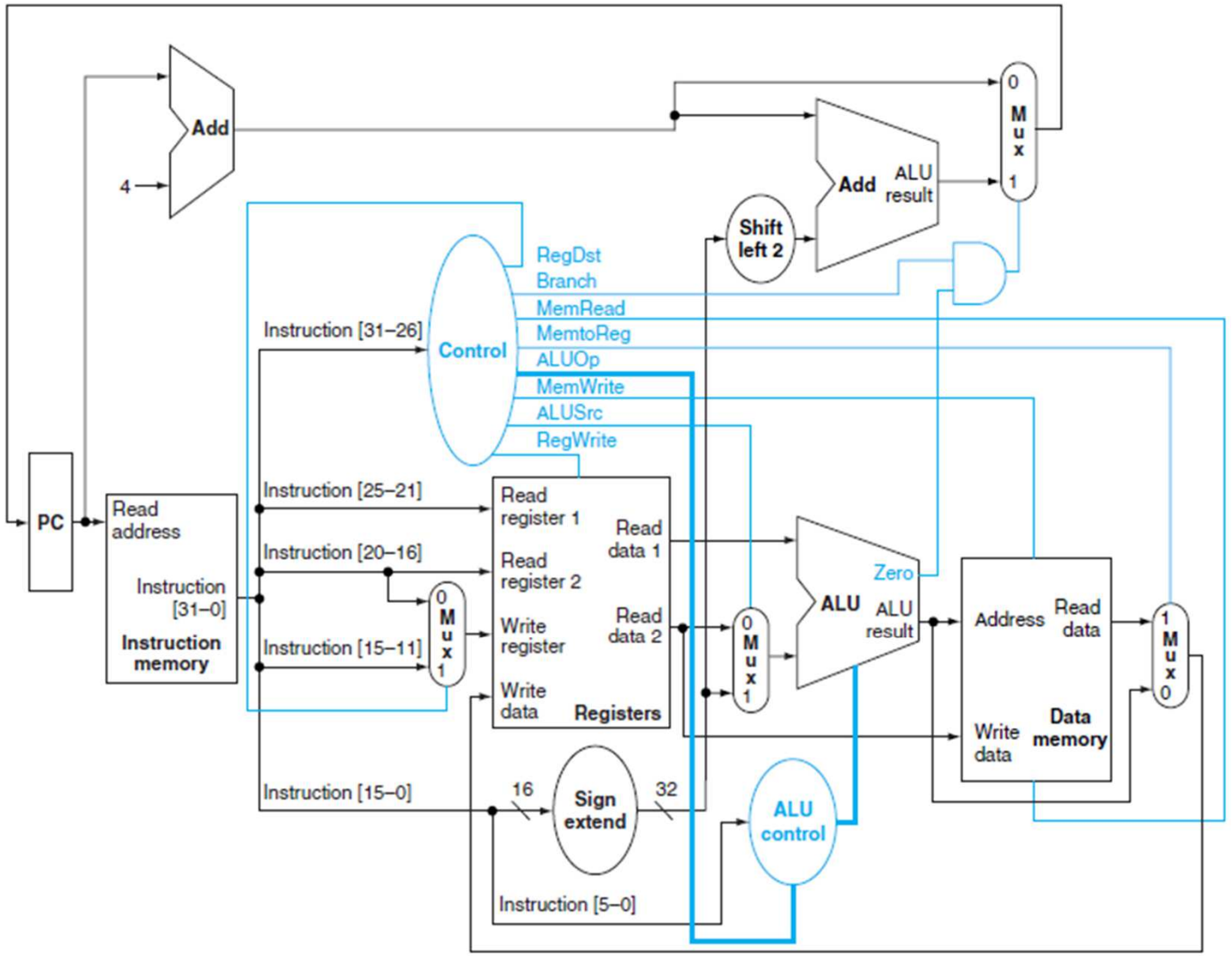
Sinal	Efeito quando =0	Efeito quando =1
RegDst	O número do reg. destino vem do campo 20:16	O número do reg. destino vem do campo 15:11
RegWrite	nenhum	O registrador na entrada Write register é escrito com o valor da entrada Write data
ALUSrc	O segundo operando da ALU vem da segunda saída da caixa de registradores (Read data 2)	O segundo operando da ALU vem da parte imediata da instrução estendida para 32 bits
PCSrc	PC recebe PC+4	PC recebe PC mais deslocamento da instrução de desvio (beq)
MemRead	nenhum	O dado da memória no endereço Address é colocado na saída Read data
MemWrite	nenhum	O dado na entrada Write data é escrito na memória no endereço Address
MemtoReg	Um resultado da ALU é enviado à caixa de registradores para ser gravado em um registrador	Um dado da memória é enviado à caixa de registradores para ser gravado em um registrador

Unidade de Controle

- Com a tabela anterior e a tabela abaixo, podemos projetar a unidade de controle.

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

O caminho de dados completo

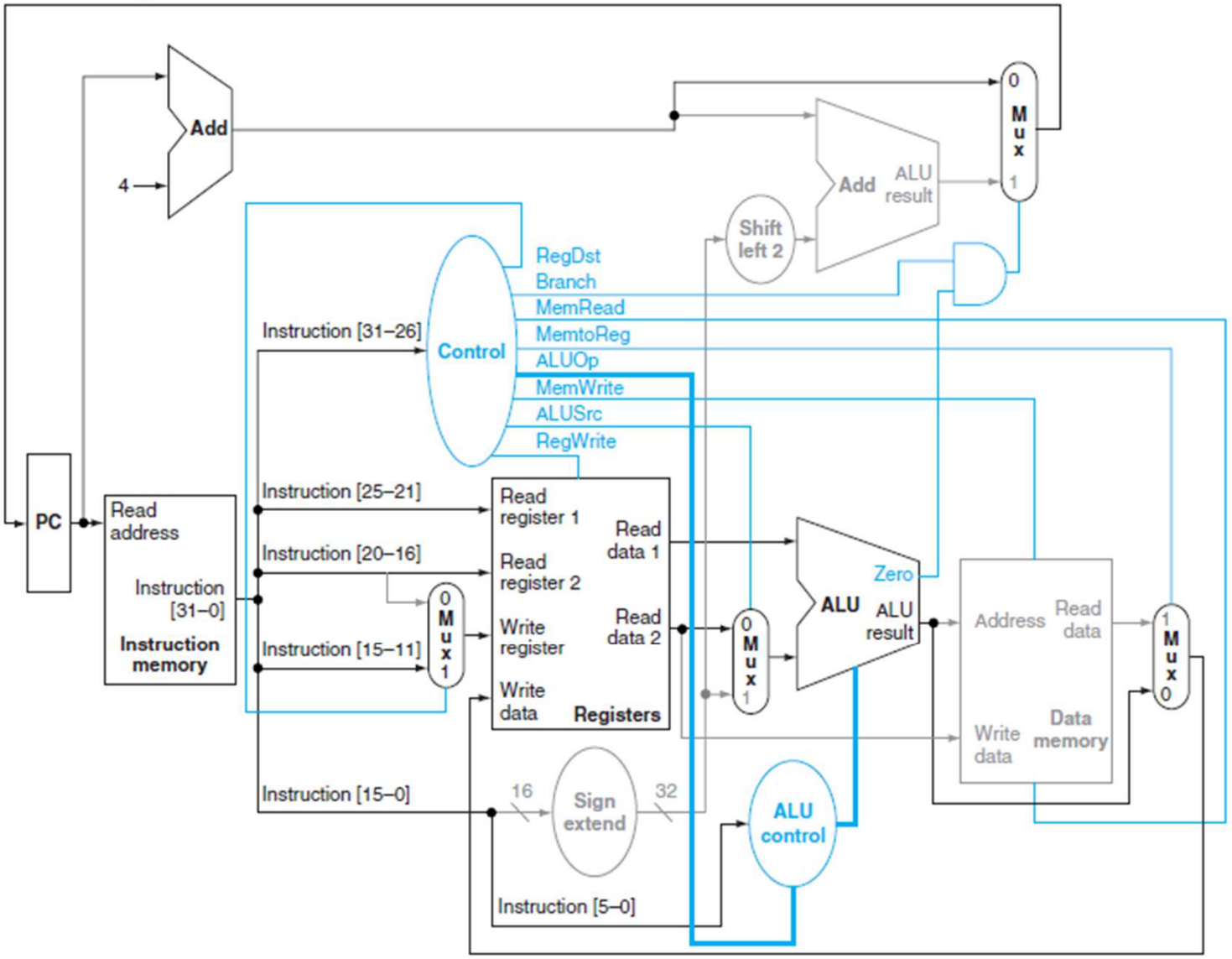


Operação do caminho de dados

- Vamos ver agora como o caminho de dados é usado em cada tipo de instrução
- Os elementos utilizados e os sinais ativos serão destacados
- A próxima figura mostra o caminho de dados para uma instrução tipo R do tipo **add $t1,$t2,$t3**.
- Acontecem 4 etapas:

1. Busca da instrução e incremento do PC
2. $t2 e $t3 são lidos no banco de registradores e os sinais de controle são definidos
3. A ALU opera os dados de entrada usando o campo funct de 6 bits (0:5) e o controle da ALU de 2 bits
4. O resultado da ALU é escrito no registrador designado pelo campo 15:11 ($t1)

Operação do caminho de dados

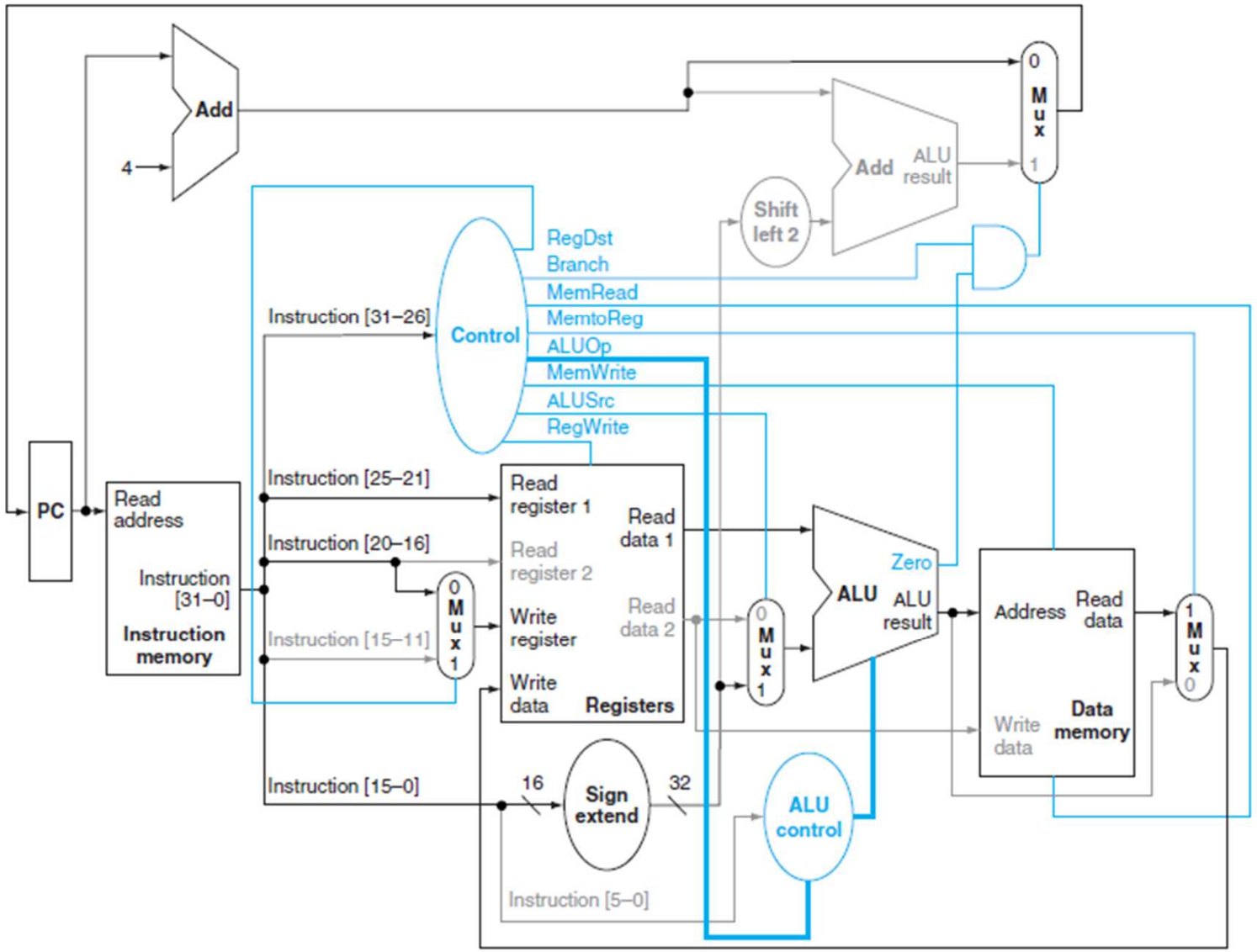


Operação do caminho de dados

- Vejamos agora uma instrução tipo **lw $t1,offset($t2)**

1. Busca da instrução e incremento do PC
2. Leitura de $t2
3. Cálculo da soma de $t2 com a parte imediata da instrução estendido de 16 para 32 bits
4. A soma da ALU é usada como endereço para a memória de dados
5. O dado da unidade de memória é escrito no banco de registradores no registrador destino fornecido pelos bits 20:16 ($t1)

Operação do caminho de dados

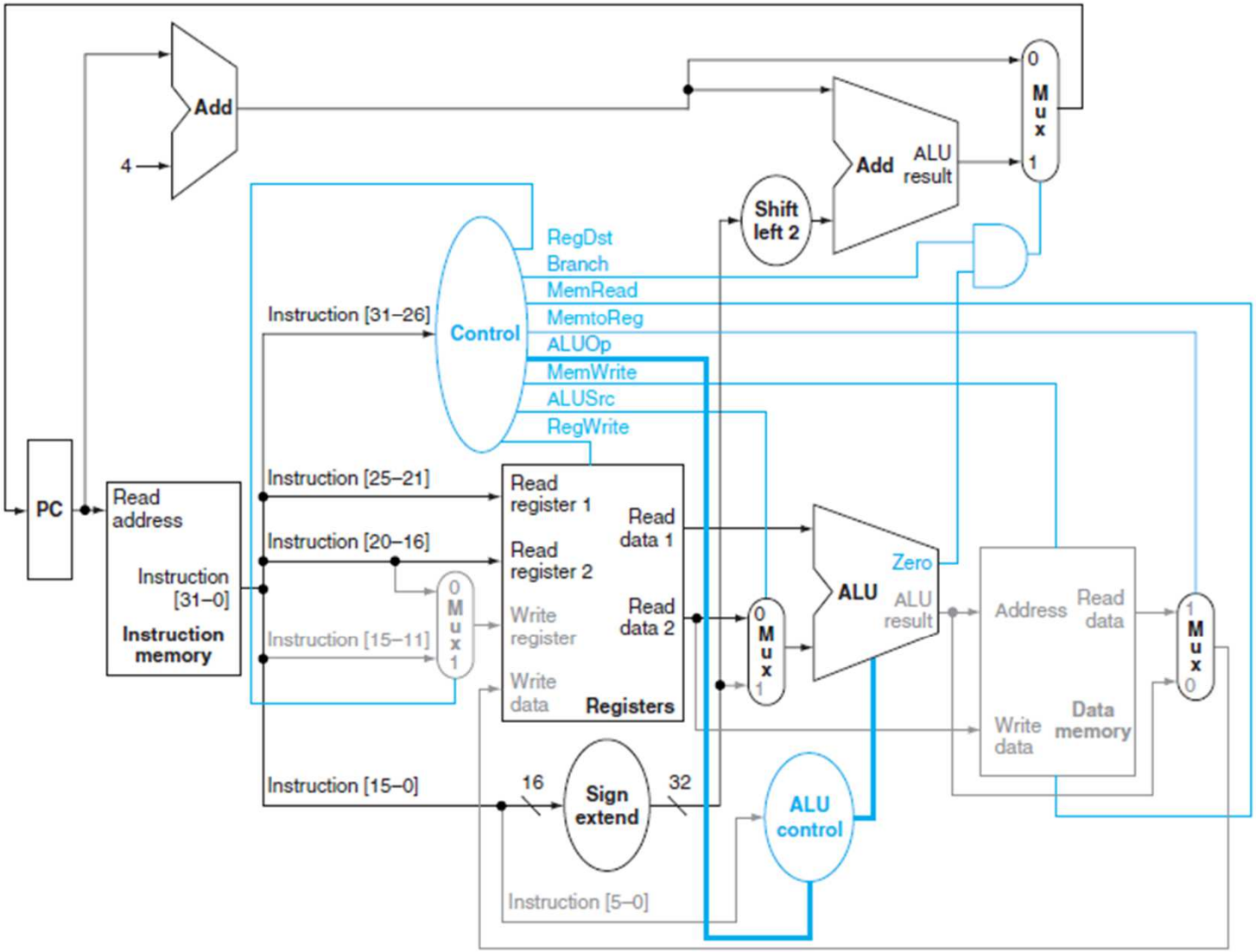


Operação do caminho de dados

- Por último vamos ver como ocorre a execução da instrução *branch if equal* (beq)

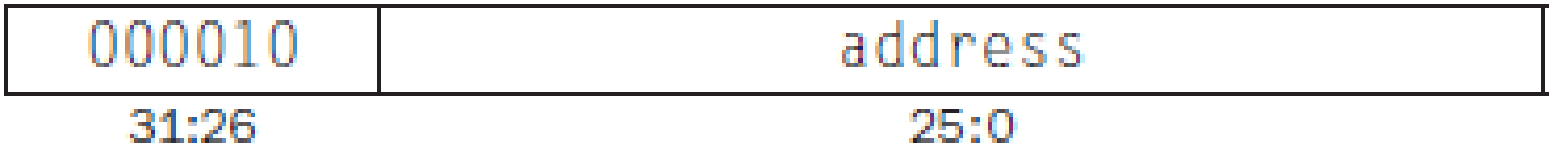
1. Busca da instrução e incremento do PC.
2. $t1 e $t2 são lidos no banco de registradores.
3. A ALU faz uma subtração com os dados de entrada. O valor de PC+4 é somado à parte imediata estendida para 16 bits gerando o endereço destino do desvio, se for tomado
4. Se a ALU gerar zero como resultado o sinal de controle gerado será nível alto, decidindo tomar o desvio. Caso contrário será usado PC+4.

Operação do caminho de dados



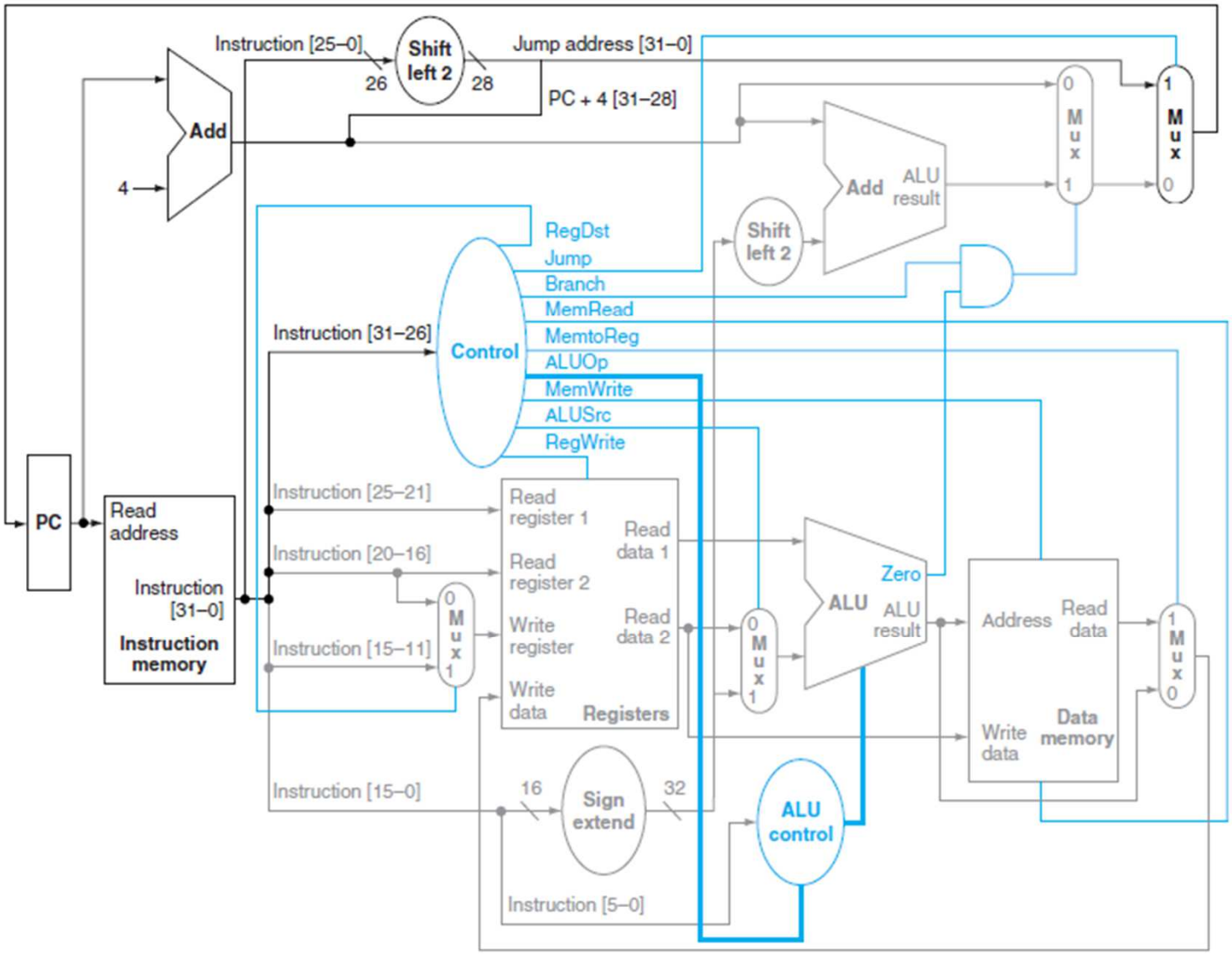
Implementando Jumps

- Os caminhos de dados mostrados não contemplam a instrução jump.
- A instrução jump possui um endereço de 26 bits e o opcode 000010:



- A instrução jump difere da beq porque é incondicional e oi cálculo do endereço é diferente.
- O endereço resultante é a concatenação de:
  - Os 4 bits superiores de PC+4
  - Os 26 bits do campo address da instrução
  - Dois bits zero (00)

Operação do caminho de dados



Operação do caminho de dados

- Por que uma arquitetura monociclo não é usada hoje?
  - Ineficiente
  - O ciclo de clock precisa ter a mesma duração para qualquer instrução, que é determinado pelo caminho mais longo do circuito
  - Instruções mais rápidas tem a mesma duração de instruções mais longas, como o load word.
  - Embora o CPI seja 1, o ciclo de clock é maior.
  - Unidades funcionais não podem ser reutilizadas para a mesma instrução

Exemplo

- Assuma que os tempos de operação para as principais unidades funcionais sejam os seguintes:
  - Unidade de memória: 200 picoseconds (ps)
  - ALU e demais somadores: 100 ps
  - Banco de registradores (leitura ou escrita): 50 ps
- Considerando que os demais componentes não possuem atraso, qual das seguintes implementações seria mais rápida:

1. Monociclo com ciclo fixo
2. Monociclo com ciclo variável, onde cada ciclo tem o tamanho mínimo necessário para cada instrução.

- Considere o seguinte mix de instruções: 25% loads, 10% stores, 45% ALU, 15% desvios e 5% jumps

Resposta

Tempo de execução da CPU	=	contagem de instruções	x CPI x	tempo de ciclo de clock

Como CPI = 1:

Tempo de execução da CPU	=	contagem de instruções	x	tempo de ciclo de clock

Precisamos apenas encontrar o tempo de ciclo de clock para as duas implementações, já que a contagem de instruções é igual nos dois casos.

Resposta

Instruction class	Functional units used by the instruction class				
R-type	Instruction fetch	Register access	ALU	Register access	
Load word	Instruction fetch	Register access	ALU	Memory access	Register access
Store word	Instruction fetch	Register access	ALU	Memory access	
Branch	Instruction fetch	Register access	ALU		
Jump	Instruction fetch				

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400 ps
Load word	200	50	100	200	50	600 ps
Store word	200	50	100	200		550 ps
Branch	200	50	100	0		350 ps
Jump	200					200 ps

- O ciclo de clock para a máquina de clock fixo é o tempo de pior caso, ou seja, 600ps
- A outra versão terá um clock mínimo de 200ps e máximo de 600ps

Resposta

- Ciclo de clock =

600x25% + 550x10% + 400x45% + 350x15% + 200x5% =

=447,5 ps

Fator de desempenho:

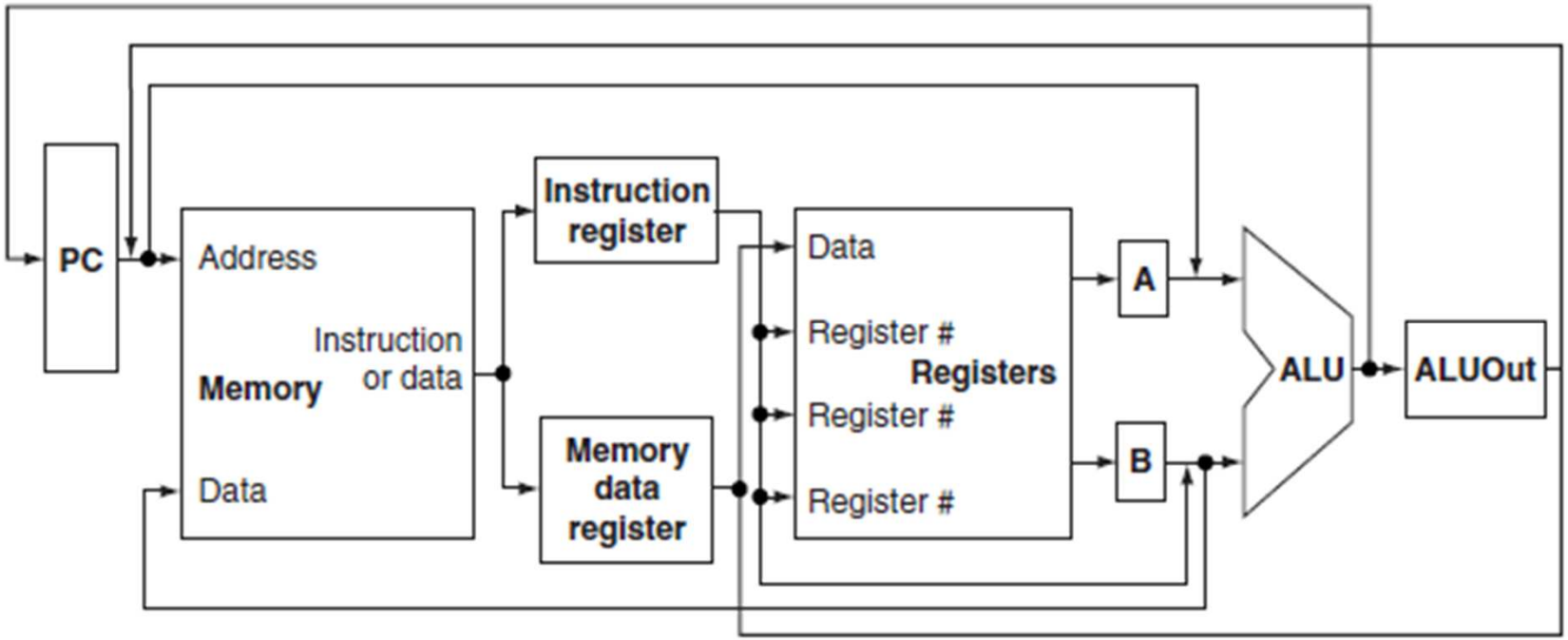
$$= \frac{600}{447,5} = 1,34$$

Implementação Multiciclo

- Modelo anterior era composto de etapas tudo usando um só ciclo
- No modelo multiciclo, cada etapa leva 1 ciclo de clock
- Permite que uma unidade funcional seja usada mais de uma vez
- Portanto, as duas principais vantagens são:

1. Instruções com diferentes CPI
2. Compartilhamento de unidades funcionais durante a execução de uma única instrução

Implementação Multiciclo

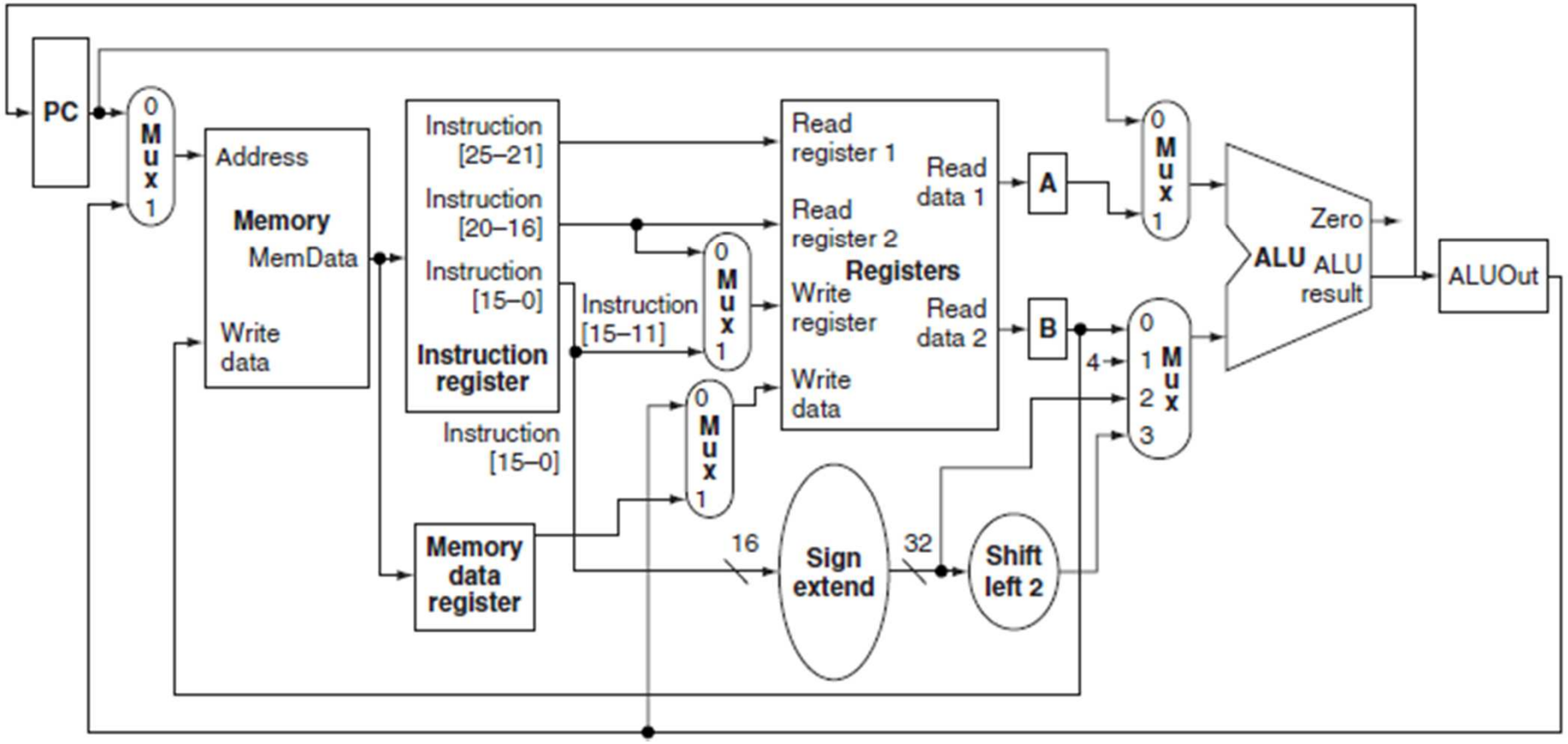


Diferenças:

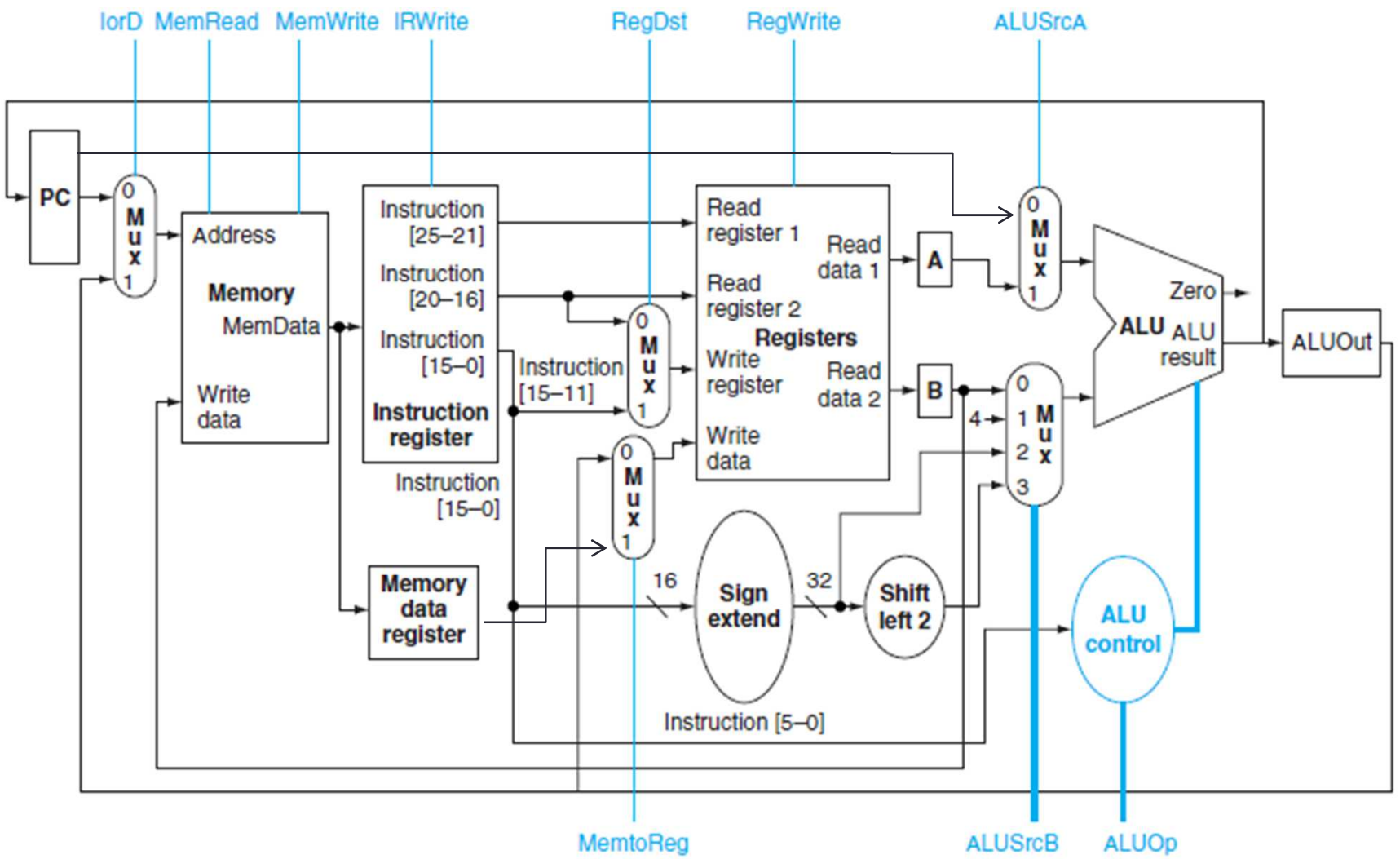
1. Uma única unidade de memória para instruções e dados
2. Uma única ALU em vez de uma ALU e dois somadores
3. Registradores intermediários entre as etapas

Implementação Multiciclo

- O caminho de dados multiciclo para o MIPS executar as instruções básicas fica:



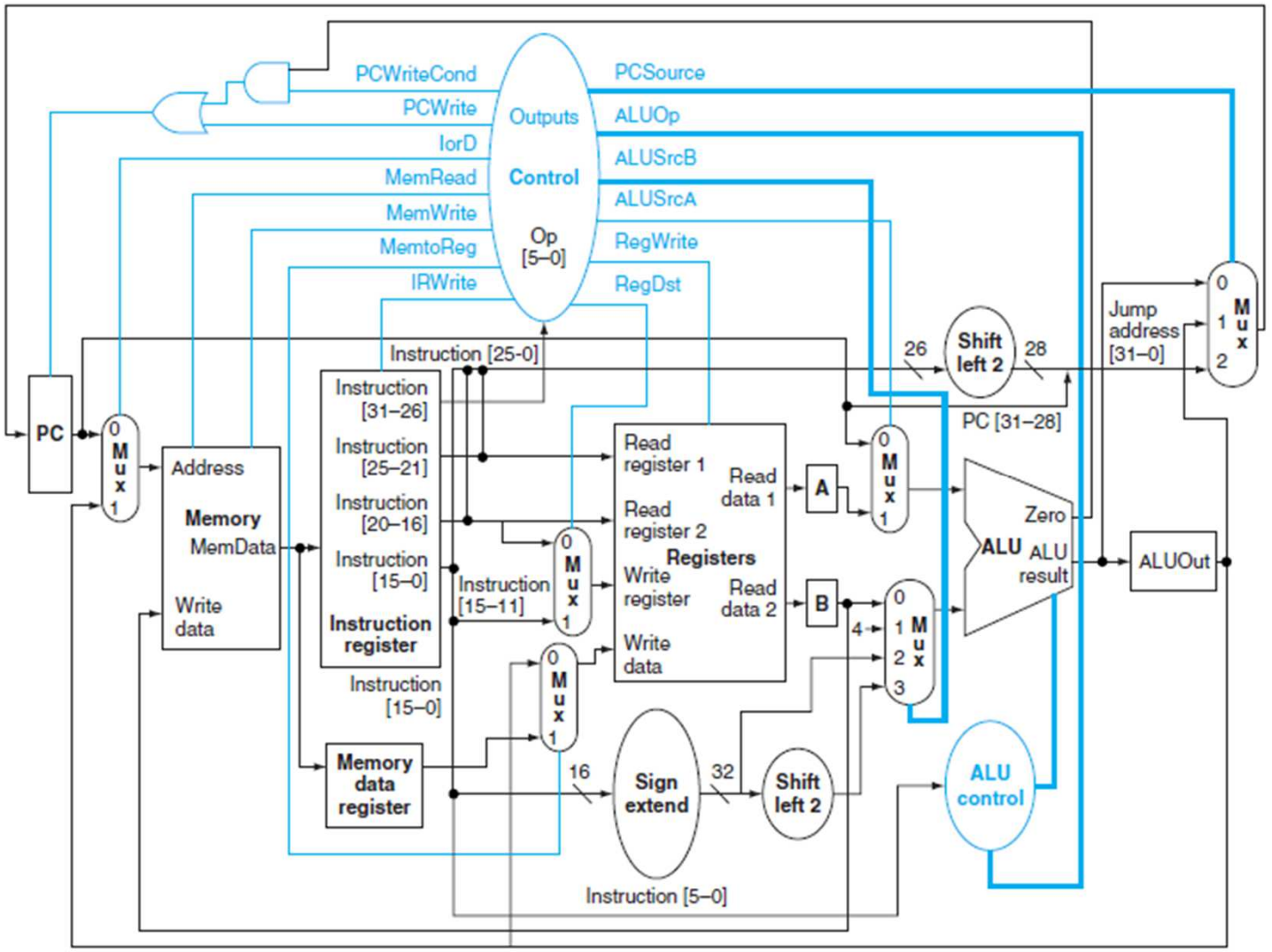
Implementação Multiciclo



Implementação Multiciclo

- Considerando que o ciclo de clock acomoda no maximo:
  - Um acesso à memória
  - Um acesso ao banco de registradores (leitura ou escrita)
  - Uma operação da ALU
- Para isso foram adicionados os registradores:
  - O registrador de instrução (IR) e registrador de dados de memória (MDR)
  - Os registradores A e B
  - ALUOut
- O modelo ainda precisa receber alguns elementos para suportar desvios e jumps
- Há três origens possíveis para o valor a ser escrito no PC:
  - A saída da ALU (PC+4)
  - O registrador ALUOut (PC+desvio)
  - Registrador IR (26 bits de endereço para o jump)

Implementação Multiciclo



Etapas de execução

- Alguns cálculos são realizados previamente, mesmo que não sejam necessários (antes de identificar a instrução)
- Ao dividir a execução da instrução em ciclos de clock, as seguintes etapas são definidas:

1. Busca da instrução

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

2. Decodificação da instrução e busca dos registradores

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

Implementação Multiciclo

3. Execução, cálculo do endereço de memória ou conclusão do desvio

- Referência à memória

```
ALUOut <= A + sign-extend (IR[15:0]);
```

- Instrução lógica ou aritmética

```
ALUOut <= A op B;
```

- Desvio

```
if (A == B) PC <= ALUOut;
```

- Jump

```
# {x, y} is the Verilog notation for concatenation of
bit fields x and y
PC <= {PC [31:28], (IR[25:0],2'b00)};
```

Implementação Multiciclo

4. Etapa de acesso à memória ou conclusão de instrução tipo R

- Referência à memória

(load) `MDR <= Memory [ALUOut];`

ou

(store) `Memory [ALUOut] <= B;`

- Instrução lógica ou aritmética (tipo R)

```
Reg[IR[15:11]] <= ALUOut;
```

5. Etapa de conclusão da leitura da memória

(load) `Reg[IR[20:16]] <= MDR;`

Implementação Multiciclo

- Resumo das operações:

	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Etapa 1	IR <= Memory[PC] PC <= PC + 4			
Etapa 2	A <= Reg [IR[25:21]] B <= Reg [IR[20:16]] ALUOut <= PC + (sign-extend (IR[15:0]) << 2)			
Etapa 3	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	if (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]],2'b00)}
Etapa 4	Reg [IR[15:11]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B		
Etapa 5		Load: Reg[IR[20:16]] <= MDR		

Exemplo – CPI em uma CPU multiciclo

Usando o mix de instruções SPECINT2000 mostrado na Figura 3.26, qual é o CPI, considerando que cada estado na CPU multiciclo exige 1 ciclo de clock?

Resposta: A figura 3.26 mostra que o mix possui 25% de loads (inclui load byte e load word), 10% de stores (store byte e store word), 11% branches (beq e bne), 2% jumps (jal e jr) e o resto (52%) instruções de ALU. Pela Figura do slide anterior, o número de ciclos de cada tipo de instrução é:

- Loads: 5
- Stores: 4
- Instruções da ALU: 4
- Branches: 3
- Jumps: 3

Exemplo – CPI em uma CPU multiciclo

Core MIPS	Name	Integer	Fl. pt.	Arithmetic core + MIPS-32	Name	Integer	Fl. pt.
add	add	0%	0%	FP add double	add.d	0%	8%
add immediate	addi	0%	0%	FP subtract double	sub.d	0%	3%
add unsigned	addu	7%	21%	FP multiply double	mul.d	0%	8%
add immediate unsigned	addiu	12%	2%	FP divide double	div.d	0%	0%
subtract unsigned	subu	3%	2%	load word to FP double	l.d	0%	15%
and	and	1%	0%	store word to FP double	s.d	0%	7%
and immediate	andi	3%	0%	shift right arithmetic	sra	1%	0%
or	or	7%	2%	load half	lhu	1%	0%
or immediate	ori	2%	0%	branch less than zero	bltz	1%	0%
nor	nor	3%	1%	branch greater or equal zero	bgez	1%	0%
shift left logical	sll	1%	1%	branch less or equal zero	blez	0%	1%
shift right logical	srl	0%	0%	multiply	mul	0%	1%
load upper immediate	lui	2%	5%				
load word	lw	24%	15%				
store word	sw	9%	2%				
load byte	lbu	1%	0%				
store byte	sb	1%	0%				
branch on equal (zero)	beq	6%	2%				
branch on not equal (zero)	bne	5%	1%				
jump and link	jal	1%	0%				
jump register	jr	1%	0%				
set less than	slt	2%	0%				
set less than immediate	slti	1%	0%				
set less than unsigned	sltu	1%	0%				
set less than imm. uns.	sltiu	1%	0%				

FIGURE 3.26 The frequency of the MIPS instructions for SPEC2000 integer and floating point. All instructions that accounted for at least 1% of the instructions are included in the table. Pseudoinstructions are converted into MIPS-32 before execution, and hence do not appear here. This data is from Chapter 2 of *Computer Architecture: A Quantitative Approach*, third edition.

Exemplo – CPI em uma CPU multiciclo

O CPI é obtido pelo seguinte:

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}} = \frac{\sum \text{Instruction count}_i \times \text{CPI}_i}{\text{Instruction count}}$$

$$= \sum \frac{\text{Instruction count}_i}{\text{Instruction count}} \times \text{CPI}_i$$

A razão

$$\frac{\text{Instruction count}_i}{\text{Instruction count}}$$

é a frequência de instruções da classe de instruções i. Portanto, podemos obter:

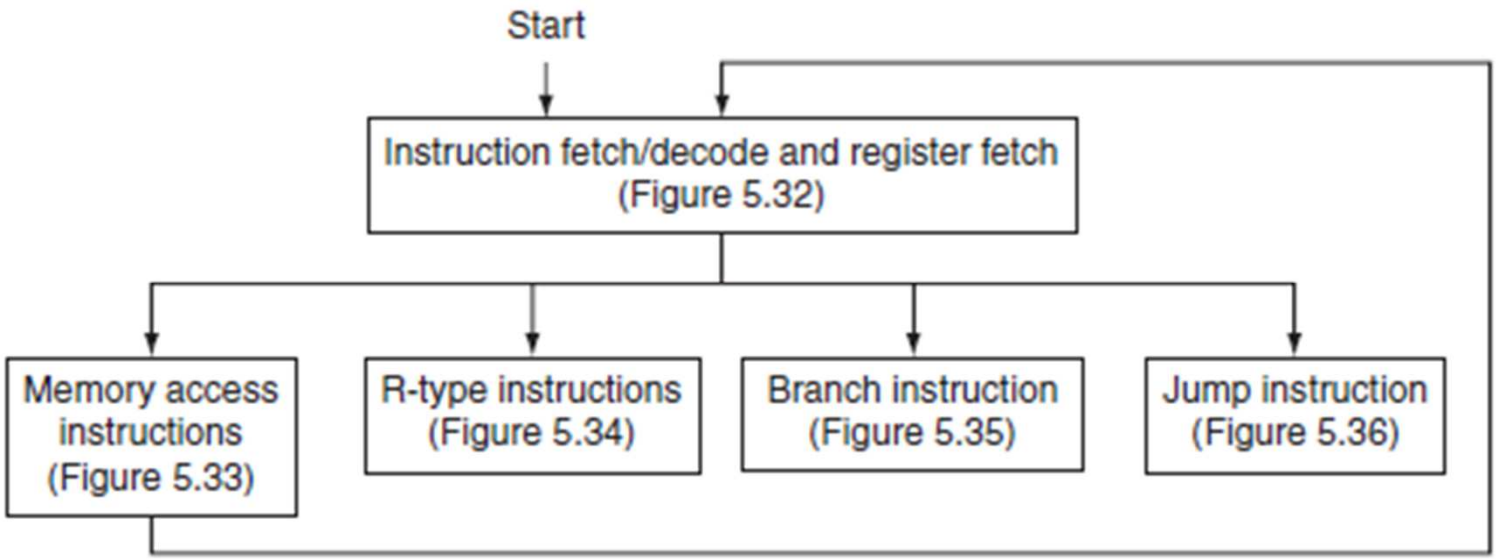
$$\text{CPI} = 0.25 \times 5 + 0.10 \times 4 + 0.52 \times 4 + 0.11 \times 3 + 0.02 \times 3 = 4.12$$

Implementação da Unidade de Controle

- Duas técnicas de implementação:
- Máquina de estados
  - É um conjunto de estados e diretrizes sobre como mudar de estado
  - Pode ser representado graficamente
  - Cada estado representa uma etapa da execução
  - É cíclica
- Microprogramação
  - É uma representação de programa para gerar o controle
  - Ambos podem ser implementados usando portas lógicas, PLAs e ROMs

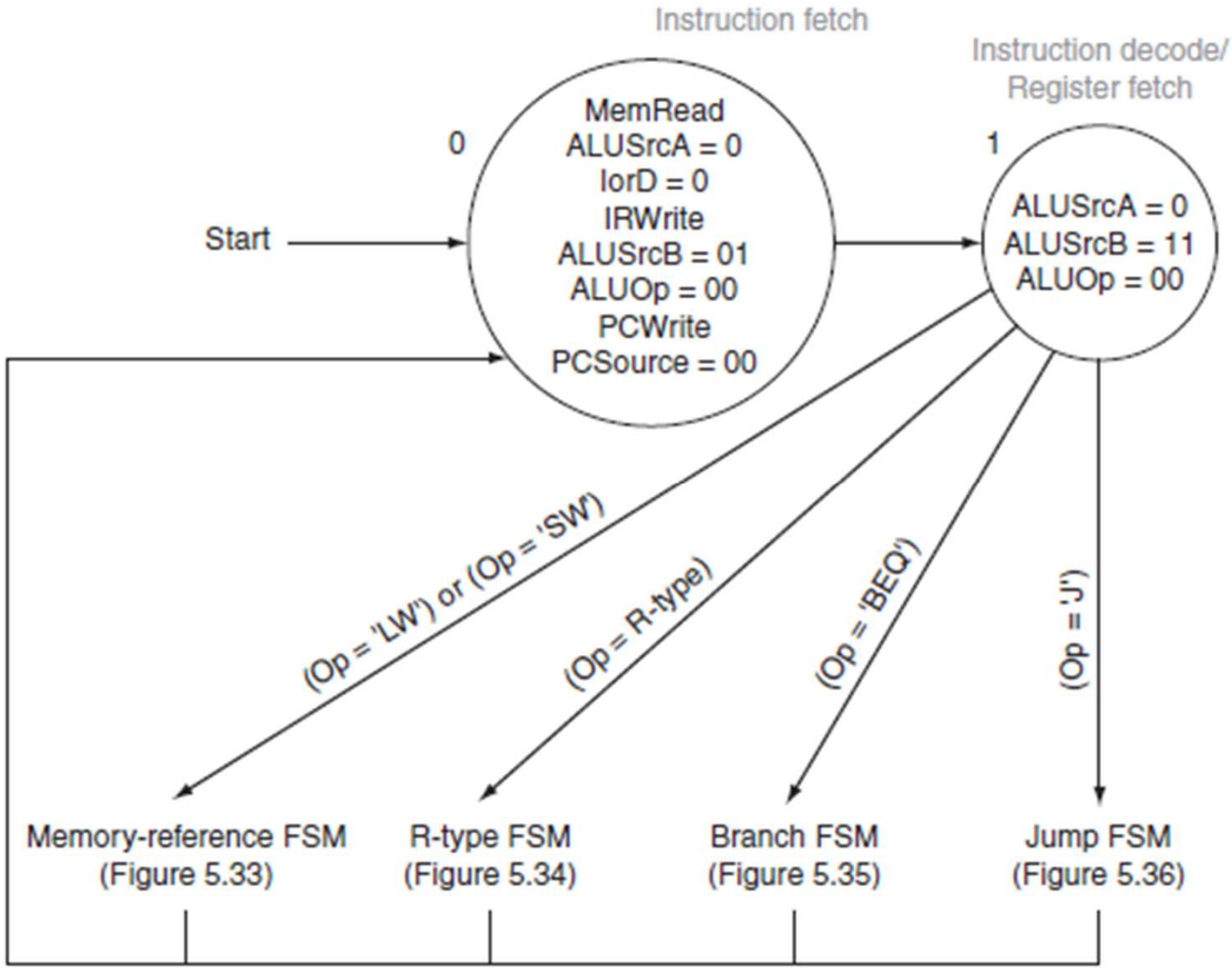
Implementação da Unidade de Controle

- O controle do MIPS corresponde basicamente às cinco etapas de execução mostradas.
- Cada estado será uma etapa e a mudança de estado ocorrerá a cada ciclo de clock.
- Uma visão abstrata é a figura:



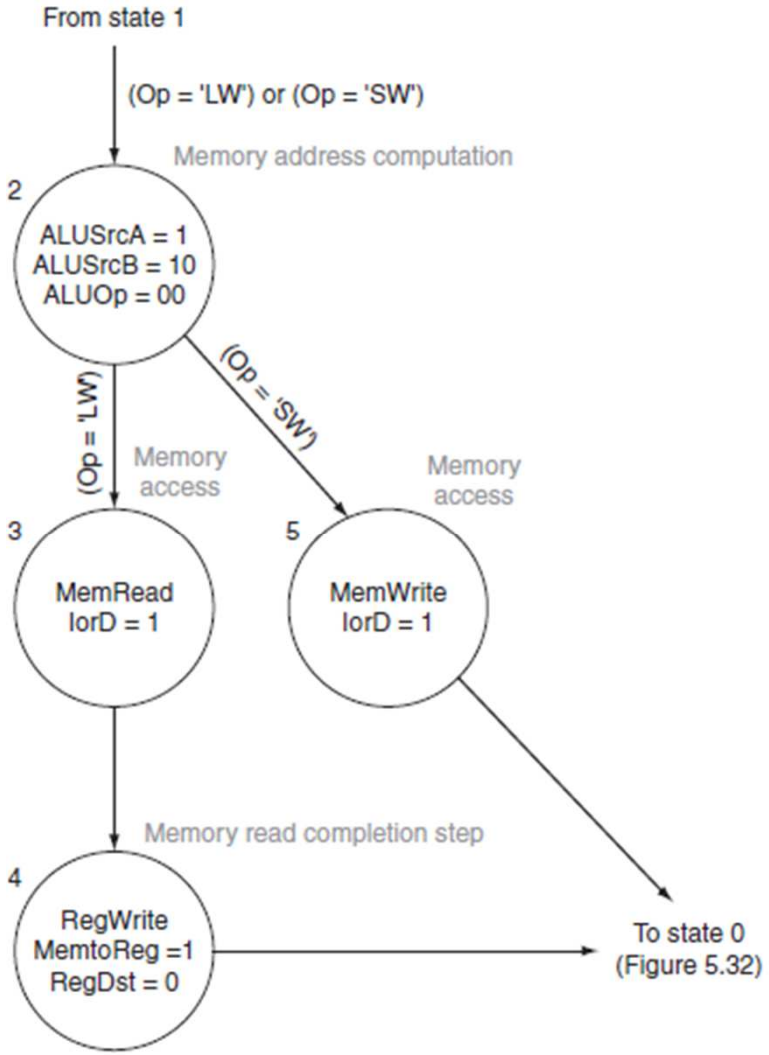
Implementação da Unidade de Controle

- Busca e decodificação da instrução:



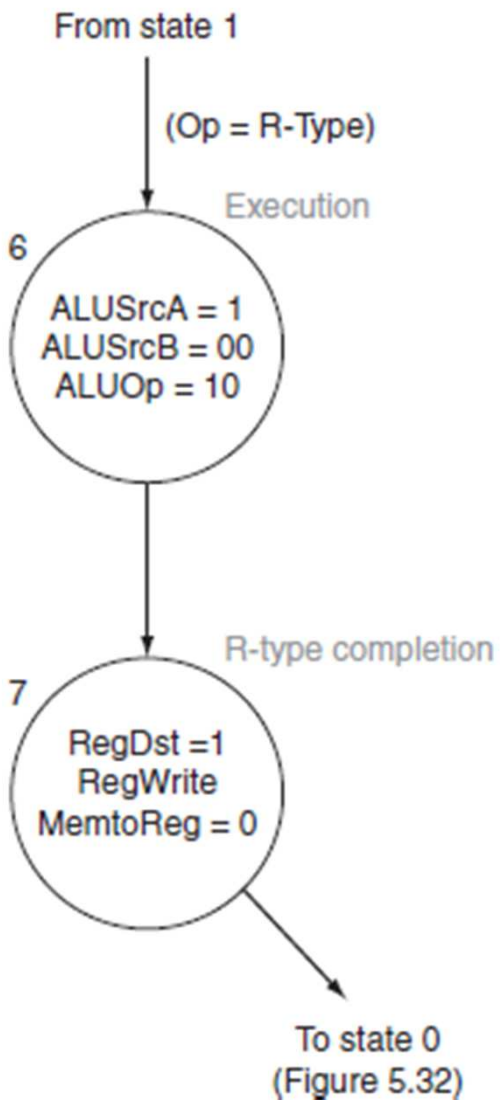
Implementação da Unidade de Controle

- Referência à memória



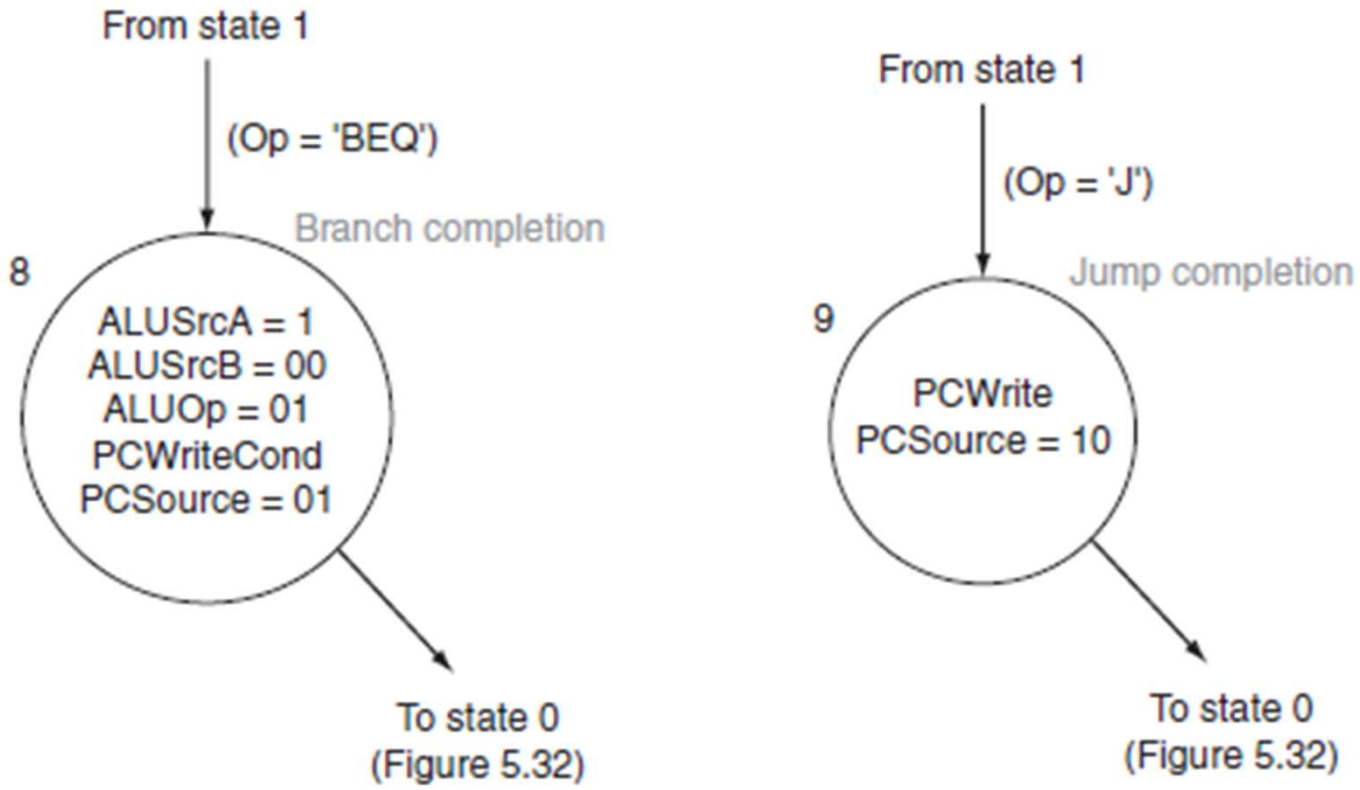
Implementação da Unidade de Controle

- Instruções tipo R



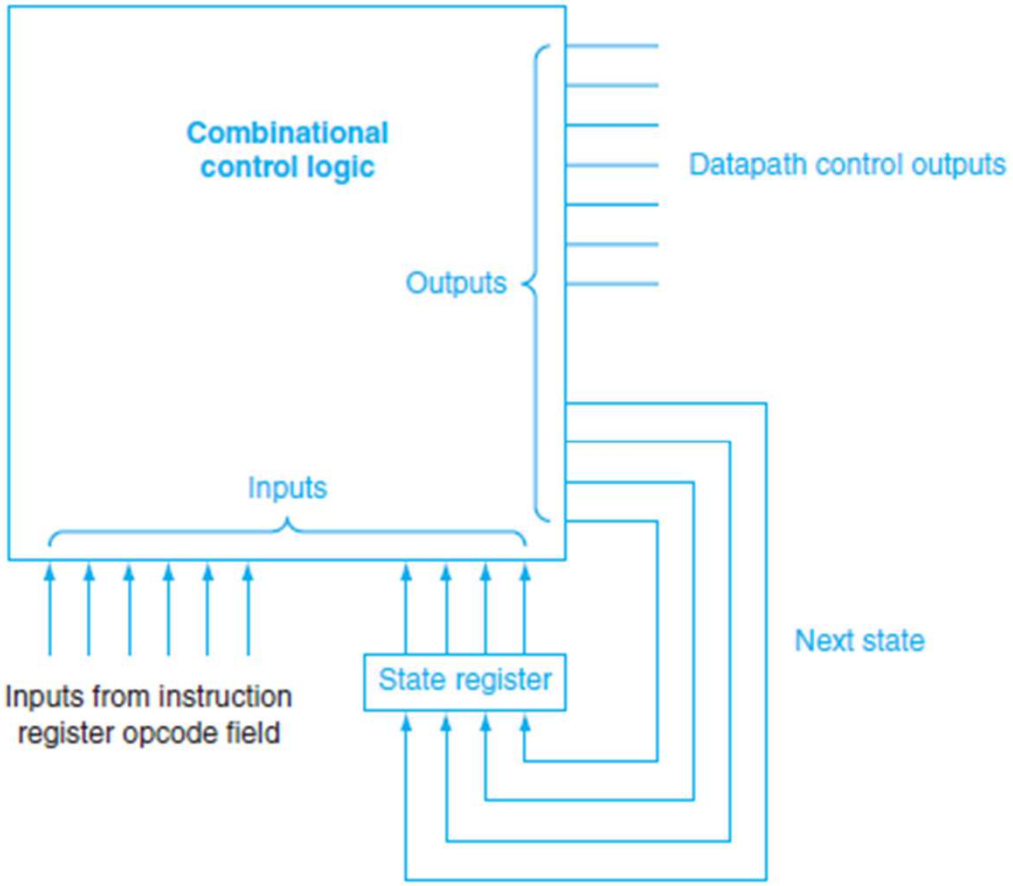
Implementação da Unidade de Controle

- Branch e jump



Implementação da Unidade de Controle

- Implementação da máquina de estados:
- Maquinas de Mealy e Moore.



Exceções

- Exceções e interrupções complicam o controle
- São eventos que não são nem jumps e nem branches que mudam o fluxo normal de execução das instruções
- Uma exceção é um evento interno, por exemplo o overflow, instrução indefinida, chamada de sistema
- Uma interrupção é um evento externo ao processador, usadas pelos dispositivos de E/S através de pinos específicos do processador
- Muitas vezes o termo interrupção é usado nos dois casos (ex.: Intel IA-32)

Exceções

- Quando ocorre uma exceção ou interrupção, o endereço da instrução atual deve ser salvo e depois transferir o controle para um endereço especificado pelo SO.
- Nessa rotina será tratado o problema. Se for um overflow a ação pode ser até a interrupção do programa e a apresentação de uma mensagem de erro
- Um método de implementação é o registrador Cause que indicará a causa da exceção.
- Outro método são as interrupções vetorizadas. Ex.:

Exception type	Exception vector address (in hex)
Undefined instruction	$C000\ 0000_{hex}$
Arithmetic overflow	$C000\ 0020_{hex}$