Aula 4: Processador - Caminho de Dados e Controle - Parte I

Professor(a): Virgínia Fernandes Mota
http://www.dcc.ufmg.br/∼virginiaferm

OCS (TEORIA) - SETOR DE INFORMÁTICA

Caminho de Dados e Controle

- Introdução
- Convenções Lógicas de Projeto
- Construindo um Caminho de Dados
- Sinais de Controle

Introdução

- Desempenho de uma máquina
  - Contagem de instruções
  - Tempo de ciclo de clock
  - Ciclos de clock por instrução (CPI)
- Estamos prontos para ver uma implementação simplificada do MIPS
  - Instruções de referência à memória: lw, sw
  - Instruções lógicas e aritméticas: add, sub, and, or, slt
  - Instruções de fluxo de controle: beq, j

Introdução

- Recordando → lw: load word; sw: store word
- slt: set on less than
  slt $s1, $s2, $s3
  if ($s2 < $s3)
  $s1 = 1;
  else
  $s1 = 0
- beq: branch on equal
  beq $s1, $s2, L
  if ($s1 == $s2)
  goto L;

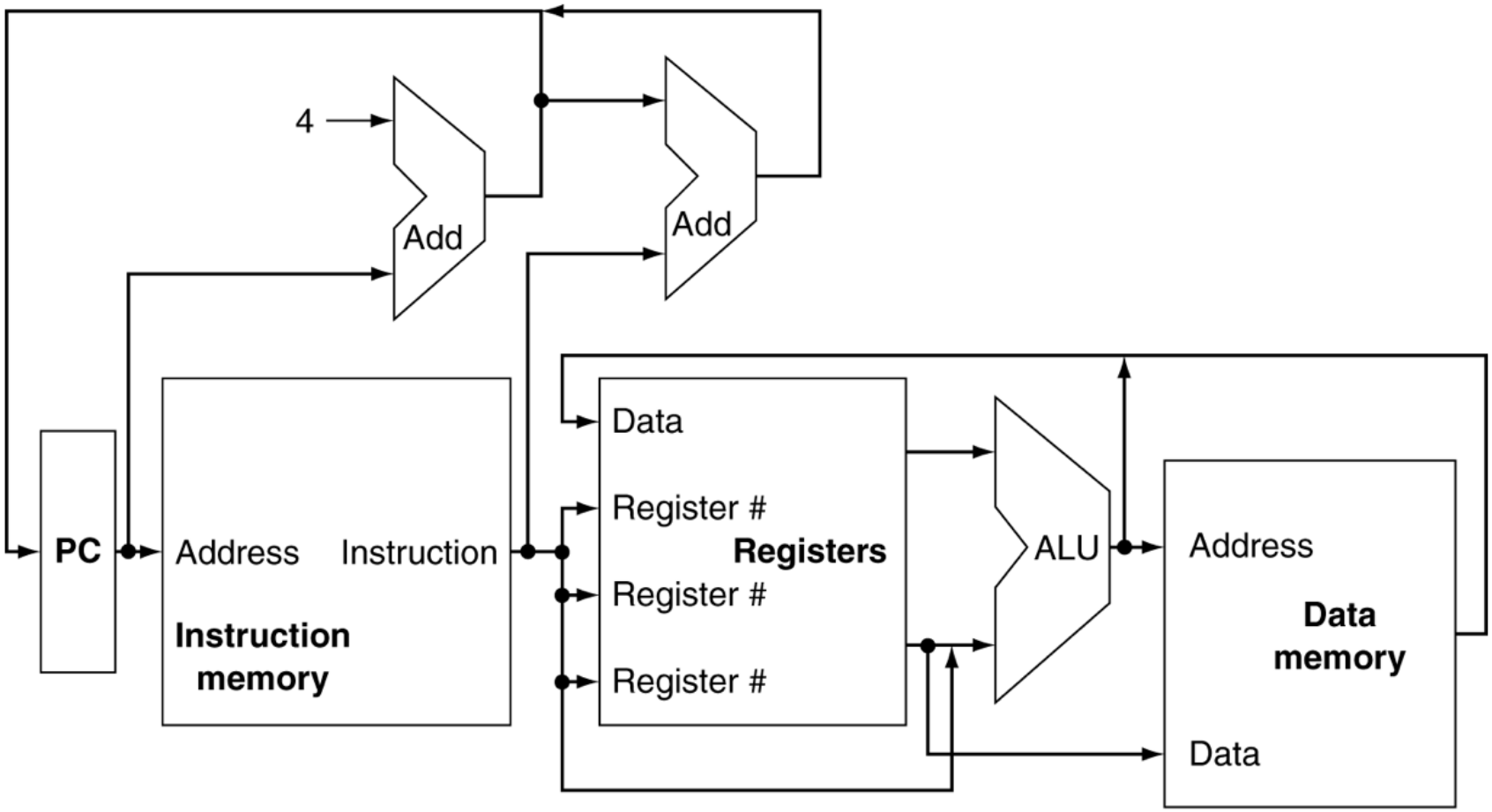
Introdução

- Implementação genérica (independente da classe da instrução):
  - Use o contador de programa (PC) para fornecer endereço de instrução
  - Obtenha a instrução da memória
  - Usando campos da instrução, leia os registradores
  - Use a instrução para decidir exatamente o que fazer
- Todas as instruções usam a ALU após lerem os registradores
  - Referência à memória: cálculo do endereço
  - Lógica/Aritmética: escrever dados da ALU para um registrador
  - Desvio: Mudar PC para incrementá-lo em 4 bytes

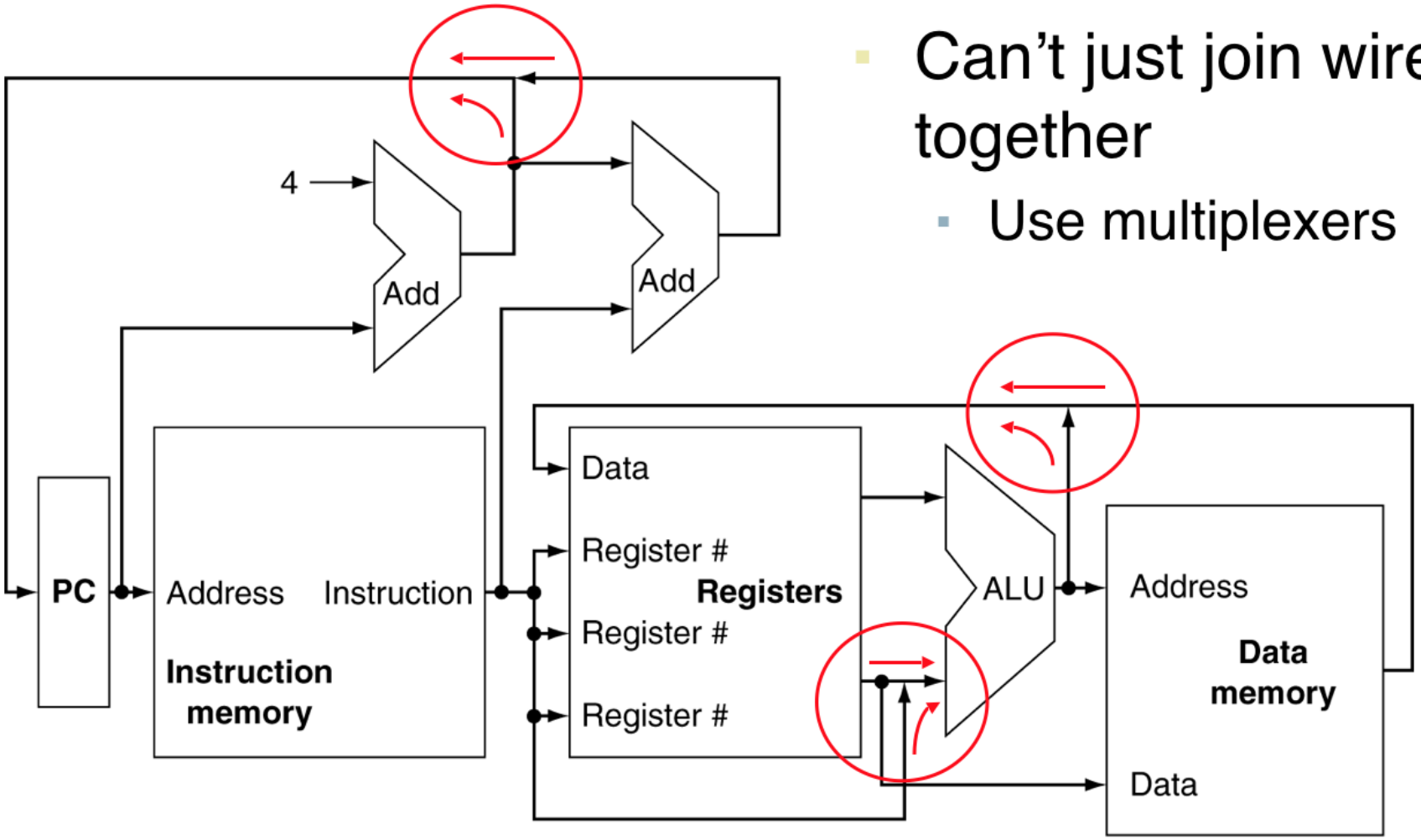
Introdução

- Dois aspectos:
  - Dados vindos de duas origens diferentes $\rightarrow$ multiplex
  - Unidades controladas de acordo com o tipo de instrução $\rightarrow$ linhas de controle

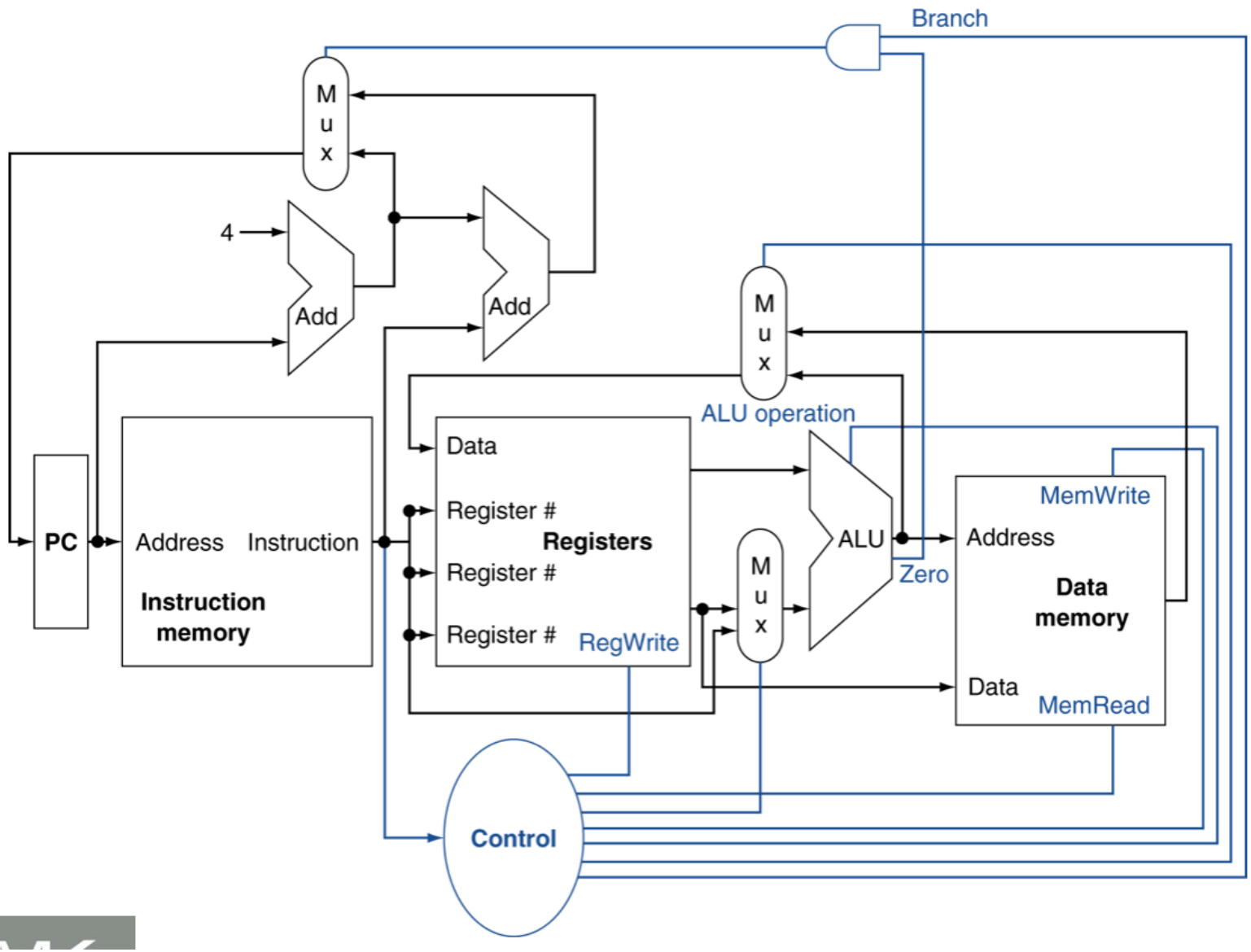
Introdução


4
Add
Add
PC
Address
Instruction
Instruction memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data memory
Data

- Can't just join wires together
  - Use multiplexers



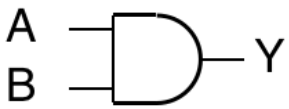
Introdução


Branch
M
u
x
4
Add
Add
M
u
x
ALU operation
PC
Address
Instruction
Instruction
memory
Data
Register #
Registers
Register #
Register #
RegWrite
M
u
x
ALU
Zero
MemWrite
Address
Data
memory
Data
MemRead
Control

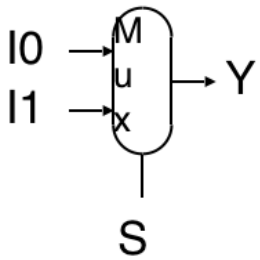
Convenções Lógicas de Projeto

- Dois tipos de unidades funcionais:
  - Elementos que operam nos valores de dados (combinacionais)
  - Saídas dependem apenas das entradas atuais
  - Elementos que contêm estado (seqüenciais)
  - Saídas dependem de suas entradas e do conteúdo do estado interno

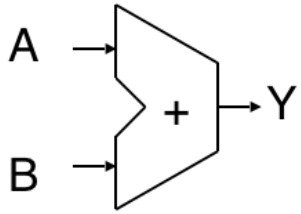
- AND-gate
  - Y = A & B



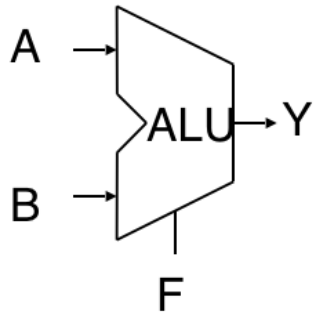
- Multiplexer
  - Y = S ? I1 : I0



- Adder
  - Y = A + B



- Arithmetic/Logic Unit
  - Y = F(A, B)

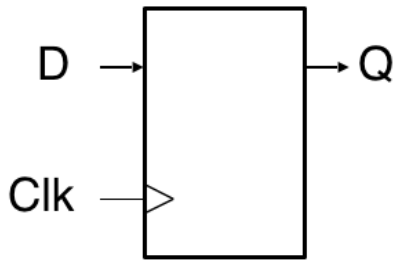


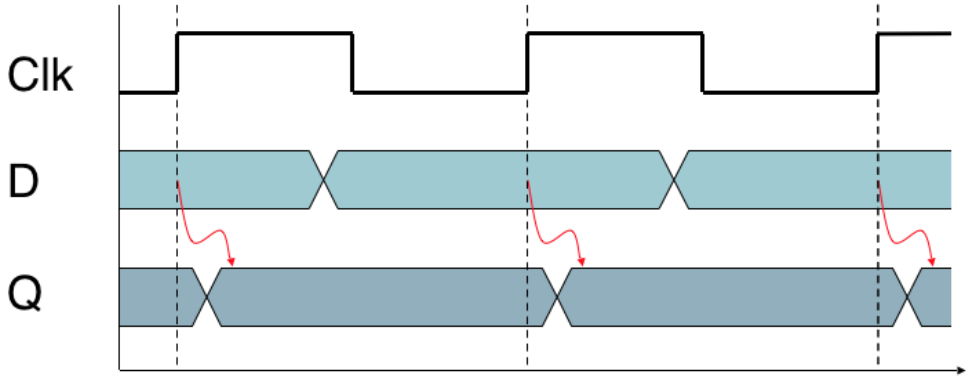
Elementos de Estado

- Um elemento com memória
- Duas entradas
  - Dado a ser escrito
  - Clock: quando dado será escrito
- Saída
  - Valor escrito em um ciclo anterior

Metodologia de Clocking

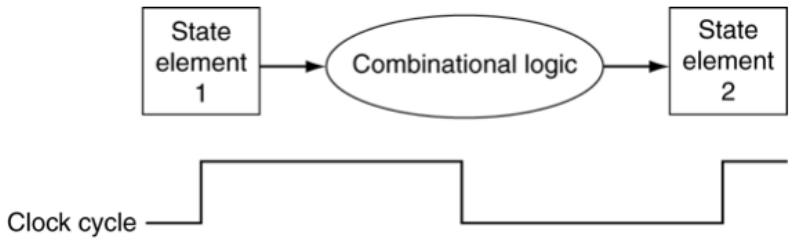
- Define quando os sinais podem ser lidos e quando podem ser escritos
  - Não desejaríamos ler um sinal ao mesmo tempo em que ele estivesse sendo escrito
- Sincronização acionada por transição
  - Mudanças de estado ocorrem em uma transição do clock





Metodologia de Clocking

- Todos os sinais precisam se propagar desde o elemento de estado 1 até o elemento 2 no tempo de um ciclo de clock
- Tempo define a duração do ciclo de clock
- Metodologia acionada por transição permite ler/escrever em um elemento de estado no mesmo ciclo de clock



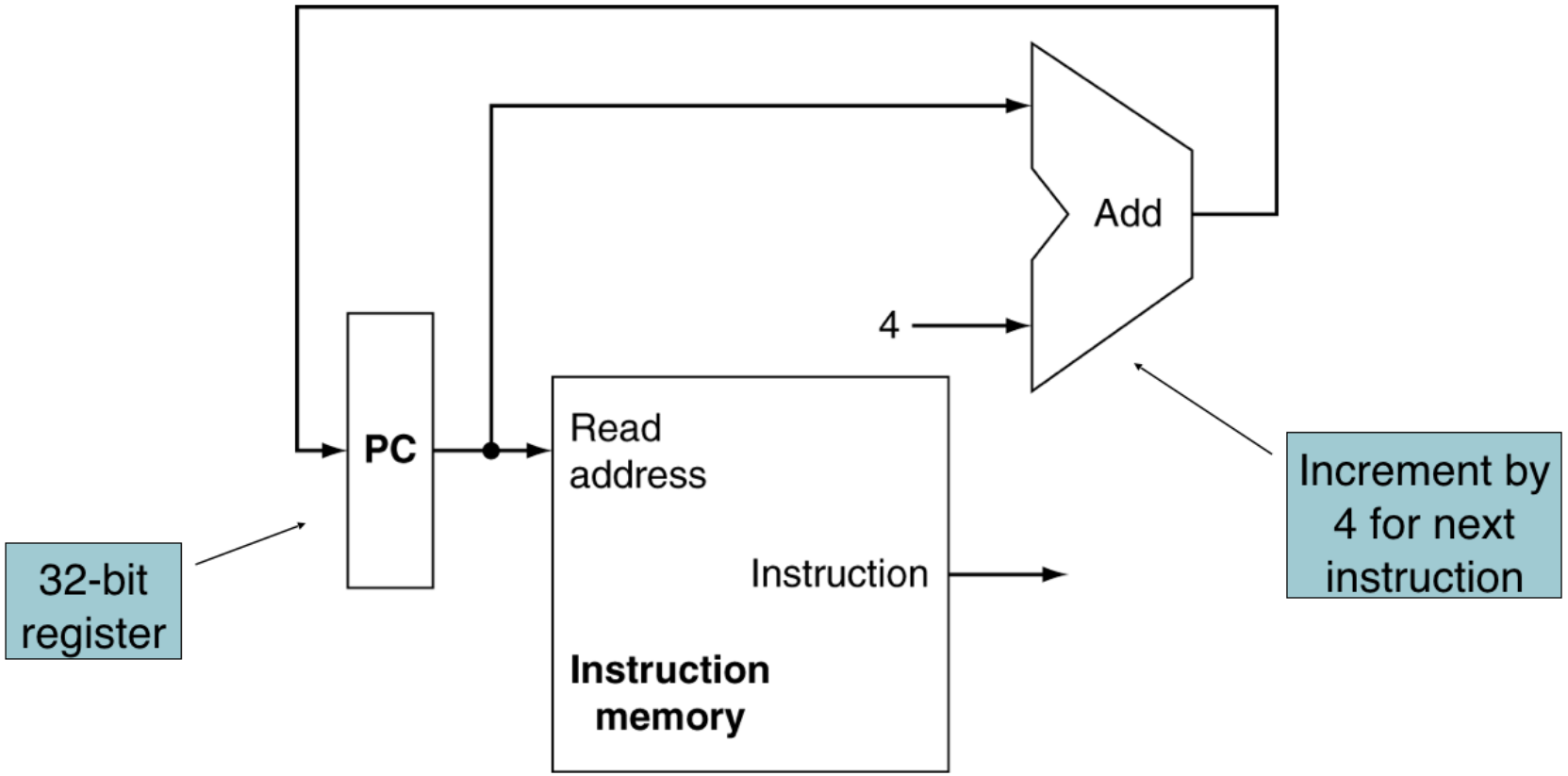
Criando um caminho de dados (*Datapath*)

- Que elementos do caminho de dados cada instrução precisa?
- Elemento do caminho de dados
  - Unidade funcional usada para operar sobre os dados ou conter esses dados dentro de um processador
  - Memória de instruções e de dados, bancos de registradores, ALU, somadores

Criando um caminho de dados (*Datapath*)

- Memória de instruções
  - Armazena instruções de um programa
  - Fornece instruções, dado um endereço
- PC
  - Registrador
  - Na arquitetura MIPS contém o endereço da instrução atual
- Somador
  - Usado para atualizar PC para o endereço da próxima instrução
- Para executar qualquer instrução
  - Busca instrução na memória
  - Para buscar a próxima, incrementa PC

Criando um caminho de dados (*Datapath*)



Criando um caminho de dados (*Datapath*)

- Instruções formato R ou instruções lógicas ou aritméticas
  - Lêem dois registradores, realizam operação na ALU com conteúdo dos registradores e escrevem o resultado
    add, sub, and, or, slt
    add \$t1, \$t2, \$t3 # \$t1 = \$t2 + \$t3
- Banco de registradores: coleção de registradores em que qualquer registrador pode ser lido/escrito especificando o número do registrador no banco

Criando um caminho de dados (*Datapath*)

- **Instruções de formato R**
- Ler duas palavras do banco de registradores
  - Entrada: número do registrador a ser lido
  - Saída: valor lido
- Escrever uma palavra no banco de registradores
  - Entradas: número do registrador e valor a ser escrito
  - Saída: conteúdo do registrador que estejam nas entradas registrador de leitura
  - Controlada pelo sinal de controle de escrita
  - Quando ativo, faz escrita na transição do clock

Criando um caminho de dados (*Datapath*)

- Banco com 32 registradores
  - 5 bits para identificar registrador
  - Entrada e saída de dados: 32 bits
- ALU
  - Entradas: dados (32 bits), e controle (4 bits, determina operação)
  - Saídas: resultado (32 bits) e zero (sinal de 1 bit quando resultado igual a zero)

a. Registers

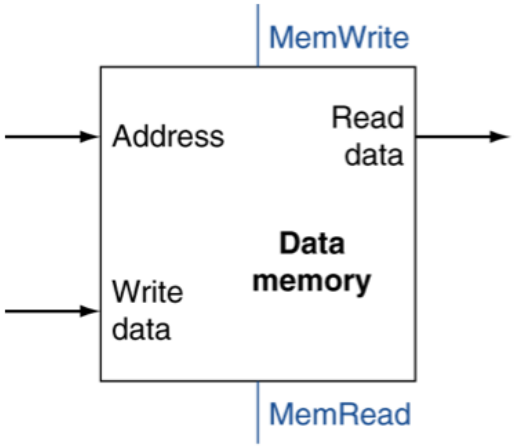
b. ALU

Criando um caminho de dados (*Datapath*)

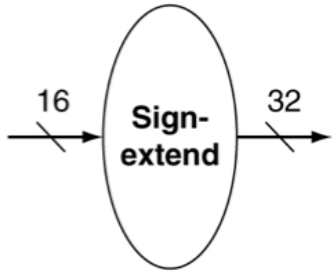
- Instruções de referência a memória
  lw $t1, offset_value($t2)
  sw $t1, offset_value($t2)
- Endereço: registrador base (t2) + offset (16 bits, contido da instrução)
  - sw: t1 armazena valor a ser escrito
  - lw: t1 indica onde valor será armazenado
- Precisamos do banco de registradores e da ALU
- Precisamos também de uma memória de dados e de uma unidade para estender o sinal do campo offset (16 para 32 bits)

Criando um caminho de dados (*Datapath*)

- Porque precisamos de um sinal de leitura e um sinal de escrita separados?
  - Ler o valor de um endereço inválido pode causar problemas



a. Data memory unit



b. Sign extension unit

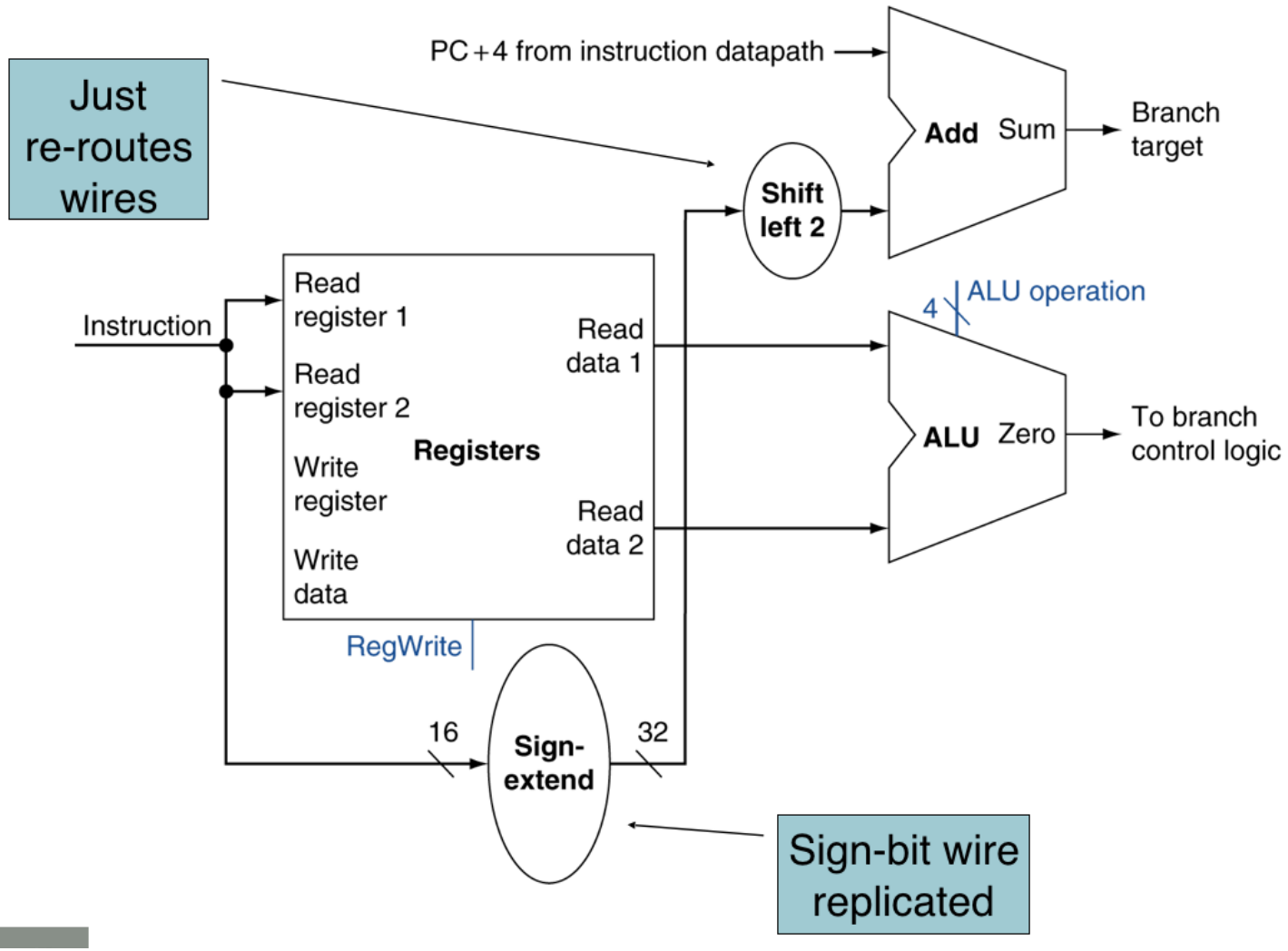
Criando um caminho de dados (*Datapath*)

- **Instrução de desvio**
  beq $t1, $t2, offset
- Offset de 16 bits: caso desvio ocorra, determina endereço do desvio em relação a instrução seguinte ao desvio
- Dois detalhes importantes:
- PC + 4 já nos dá instrução seguinte ao desvio
  - MIPS determina que offset deve ser de uma word
  - Necessário fazer deslocamento de 2 bits a esquerda

Criando um caminho de dados (*Datapath*)

- Subtração de $t1 e $t2 indica se valores são iguais
  - Zero da ALU ativo indica que sim
- Precisamos também determinar se o desvio é o não tomado
  - Desvio tomado (operadores $t1 e $t2 iguais): PC se torna endereço do desvio
  - Desvio não tomado (operadores $t1 e $t2 diferentes): PC atual incrementado

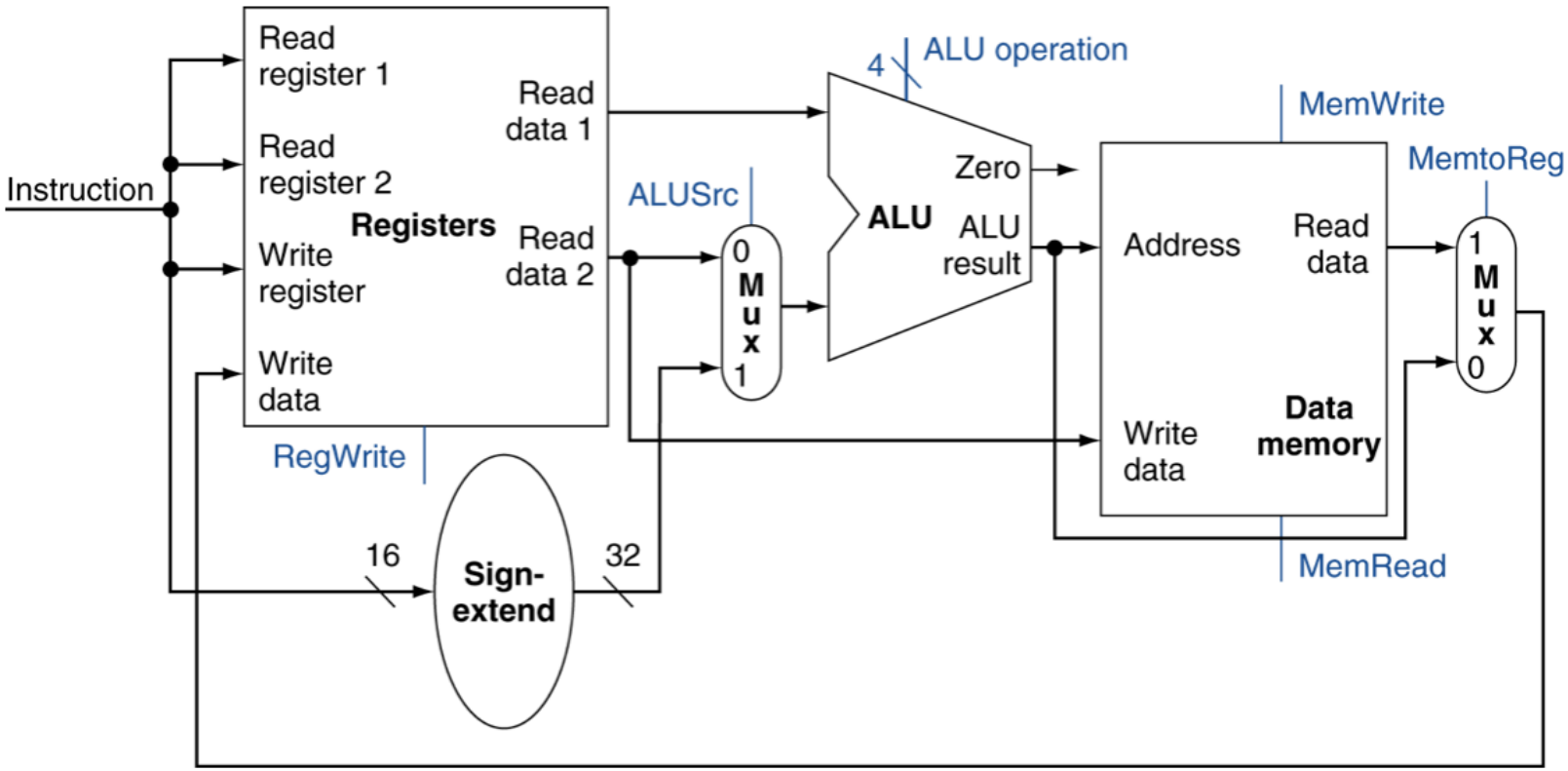
Criando um caminho de dados (*Datapath*)



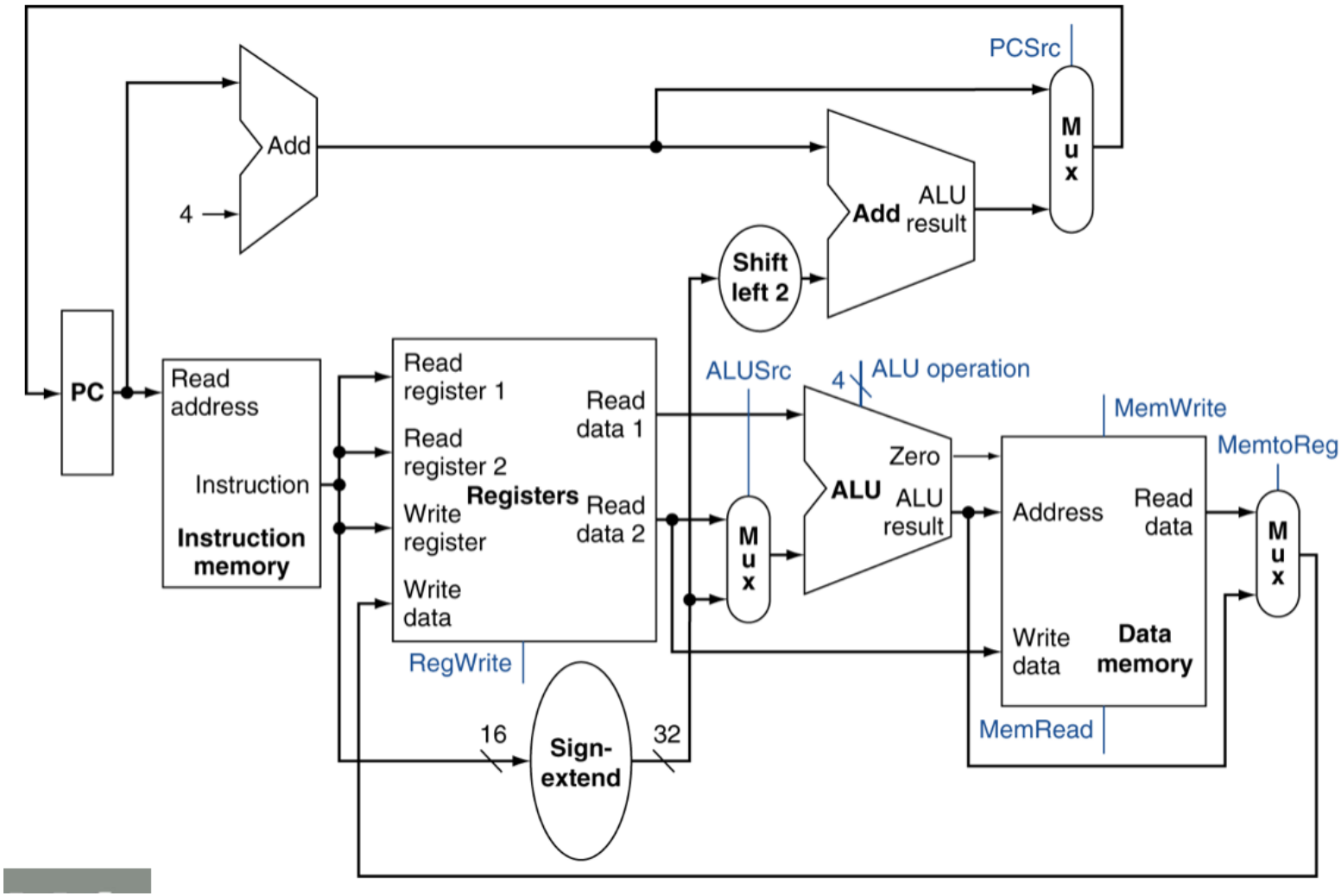
Criando um caminho de dados (*Datapath*)

- **Salto incondicional (jump)**
- Substitui 28 bits menos significativos de PC por 26 bits menos significativos da instrução deslocado 2 bits à esquerda
- Em MIPS os desvios são atrasados (delayed)
  - Instrução imediatamente seguinte ao desvio sempre é executada, independente da condição do desvio
  - Vamos ignorar esta característica neste momento

Caminho de Dados Tipo R/Load/Store



Caminho de Dados Simples



O controle da ALU

- Quatro entradas de controle
- Usaremos aqui apenas 6 das 16 combinações possíveis
- NOR usado para outras partes do conjunto de instruções

Controle	Função
0000	AND
0001	OR
0010	ADD
0110	SUB
0111	SLT
1100	NOR

O controle da ALU

- Instruções lw e sw
  - Endereço de memória calculado por adição
- Instruções tipo R (registrador)
  - Uma das cinco ações, dependendo do campo funct
- Instrução beq
  - Realizar subtração

Unidade de controle da ALU

- Entrada: campo funct da instrução e campo control, de 2 bits (OpALU)
- OpALU indica operação
  - Add (00), para load/store
  - Sub (01), para beq
  - Determinada por funct (10), instruções tipo R
- Saída: sinal de 4 bits, que controla a ALU, segundo a tabela anterior

O controle da ALU

Opcode	OpALU	Operação da Instr.	Campo *funct*	Ação Desejada	Entrada de Controle
LW	00	Load word	XXXXXX	Add	0010
SW	00	Store word	XXXXXX	Add	0010
BEQ	01	Branch equal	XXXXXX	Subtract	0110
TIPO R	10	Add	100000	Add	0010
TIPO R	10	Subtract	100010	Subtract	0110
TIPO R	10	And	100100	And	0000
TIPO R	10	Or	100101	Or	0001
TIPO R	10	Set on less than	101010	SLT	0111

O controle da ALU

- Vários níveis de decodificação!
- Unidade de controle principal gera os bits de OpALU
- Controle da ALU usa OpALU para gerar os sinais reais que controlam ALU
- Pode reduzir o tamanho da unidade de controle principal
- Pode aumentar a velocidade da unidade de controle

O controle da ALU

- Como mapear OpALU (2 bits) e funct (6 bits) para bits de controle da ALU?
- Pequeno número dos 64 valores possíveis de funct de interesse
- Campo funct apenas usado quando OpALU igual a 10
- Pequena lógica para reconhecer valores possíveis e gerar controle da ALU

O controle da ALU

OpALU		Campo funct						Operação
OpALU1	*OpALU2*	*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

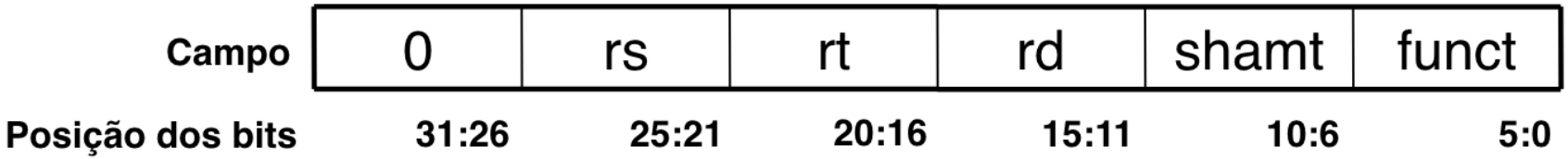
O controle da ALU

- X: don't care (Não importa)
- Saída não depende do valor de entrada correspondente a essa coluna
- Dada a tabela verdade, sua otimização e a construção das portas lógicas segue um processo mecânico
- Feitas com uso de ferramentas de CAD

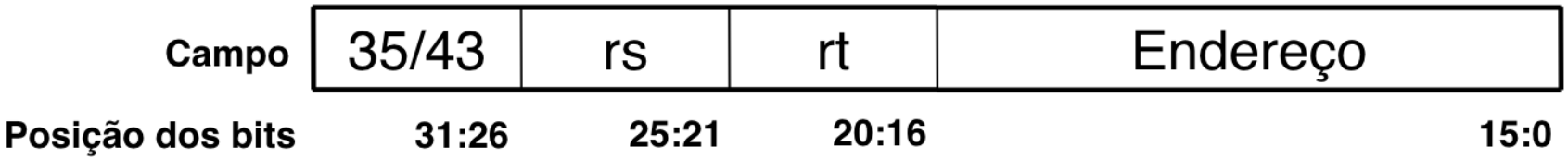
O controle da ALU

- Voltamos a considerar o caminho de dados
- Identificar campos e instruções necessários para o caminho de dados
- Como conectar os campos de uma instrução com o caminho de dados?
- Devemos examinar o formato das instruções

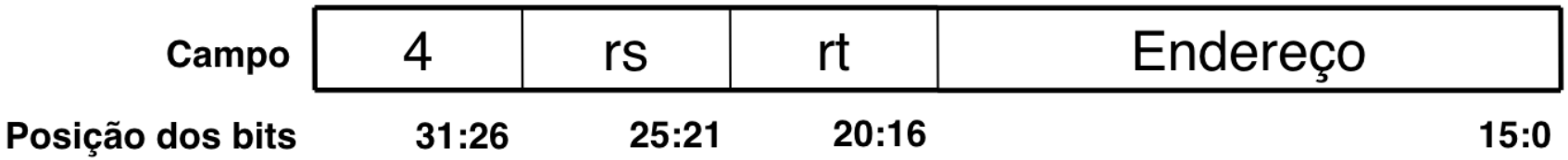
O controle da ALU



(a) Instrução tipo R



(b) Instruções load/store



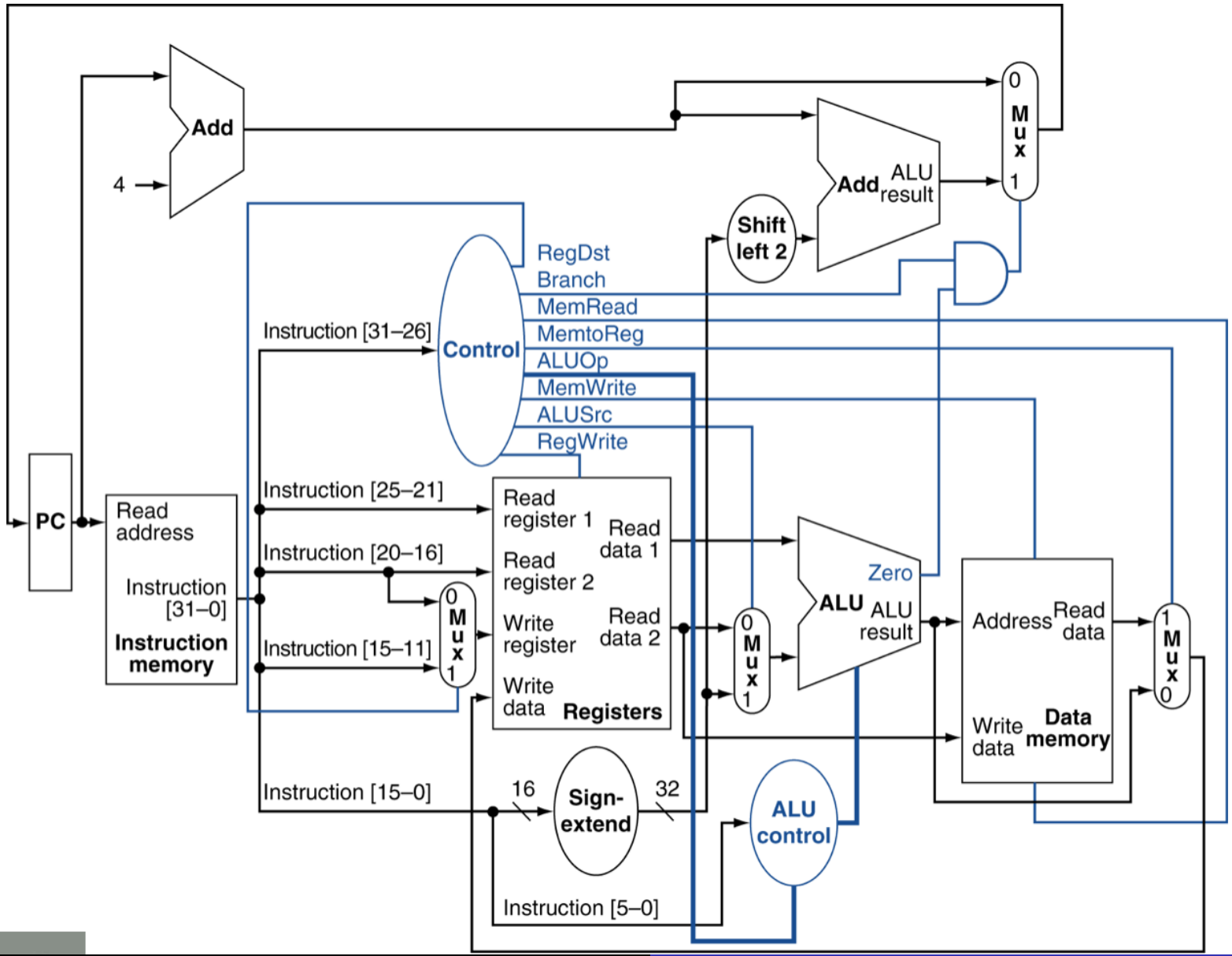
(c) Instrução beq

O controle da ALU

- Campo opcode sempre está contido nos bits 31:26. Vamos nos referir a ele como Op[5:0]
- Os dois registradores a serem lidos sempre são especificados pelos campos rs e rt, nas posições 25:21 e 20:16.
- O registrador base para as instruções load e store está sempre na posição 25:21 (rs)
- O offset de 16 bits para beq, lw e sw está sempre na posição 15:0
- Registrador de destino pode estar em dois lugares:
  Load: 20:16 (rt)
  Tipo R: 15:11 (rd)
- Precisaremos de um multiplexador

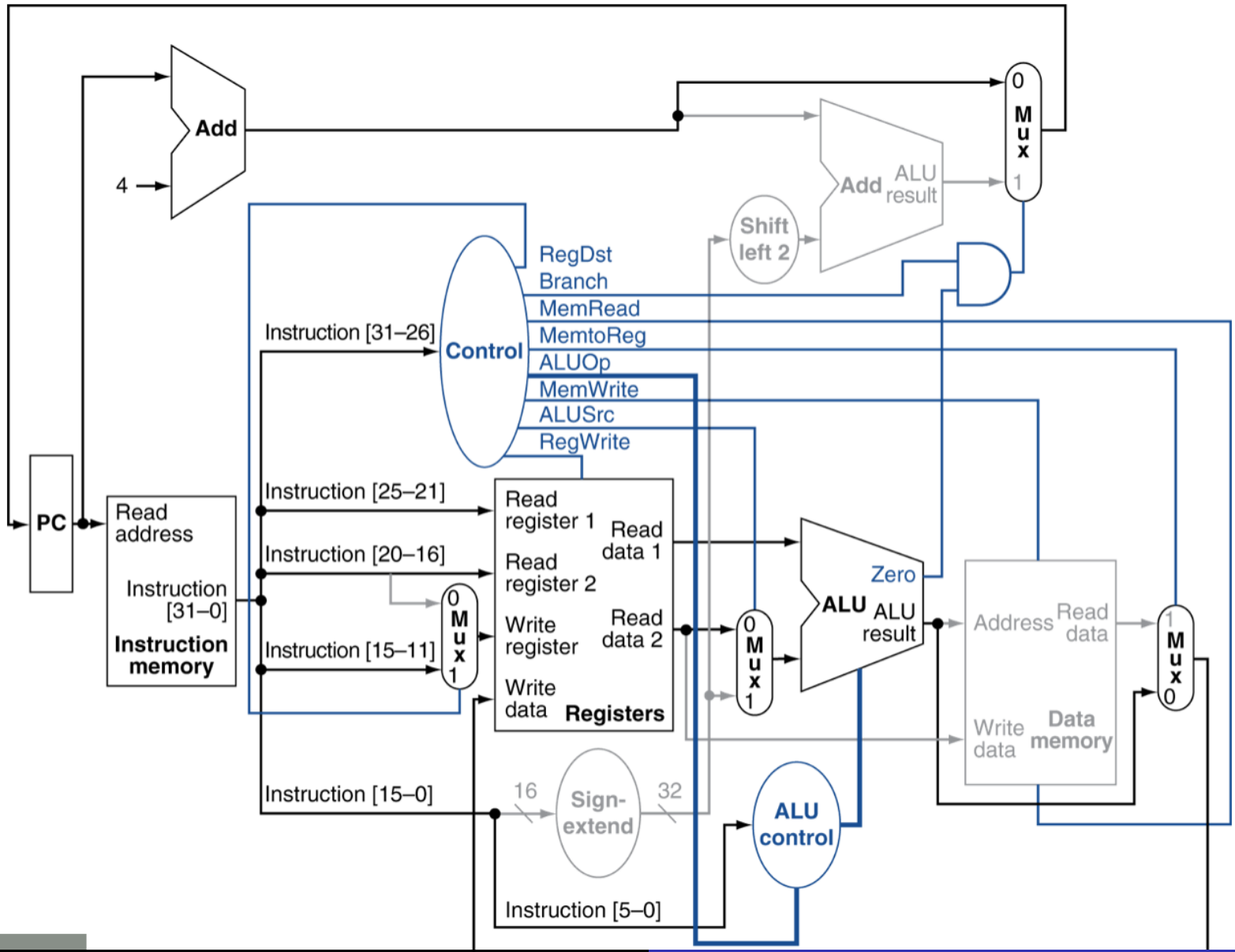
O controle da ALU

Caminho com Controle



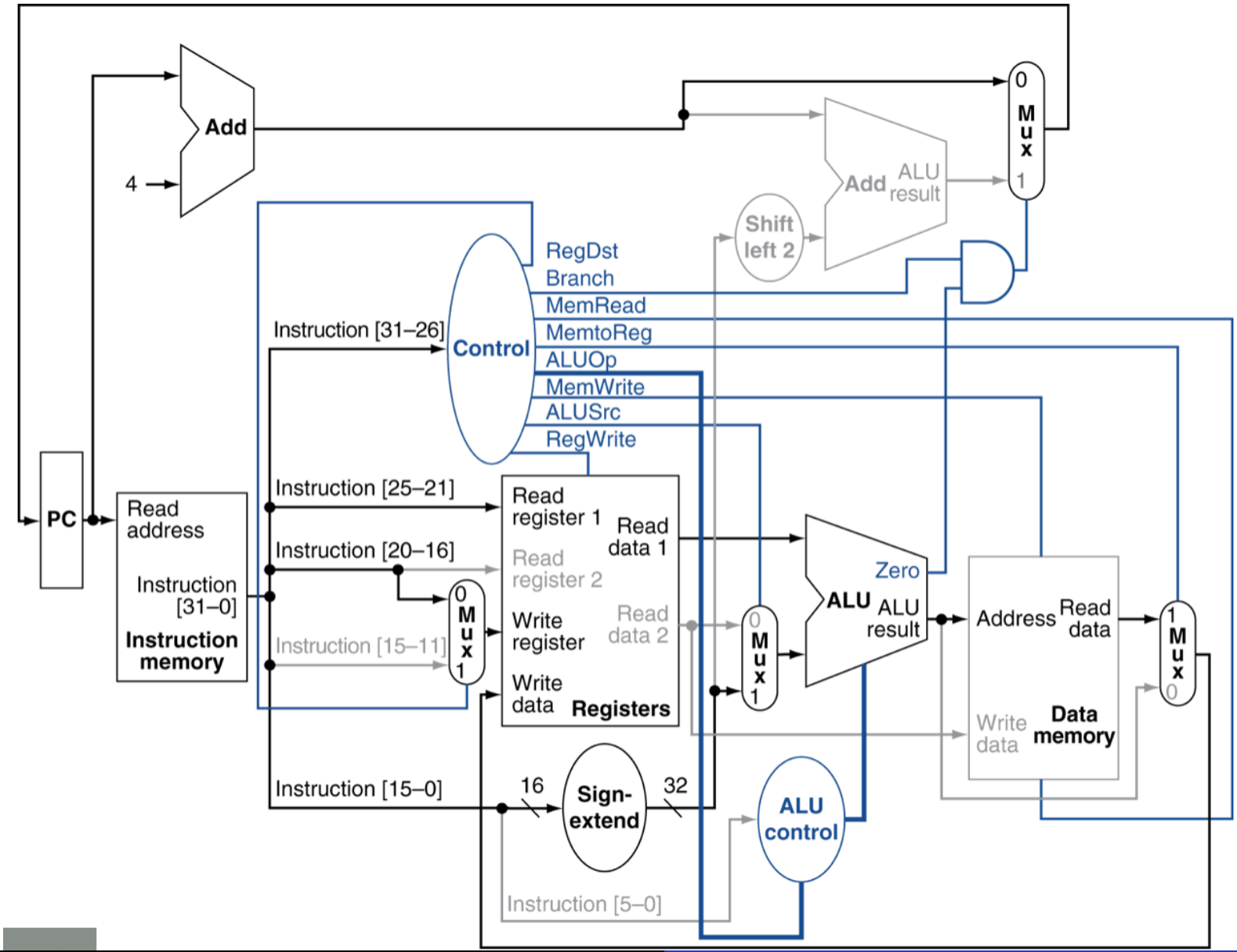
O controle da ALU

Caminho com Controle - Tipo R

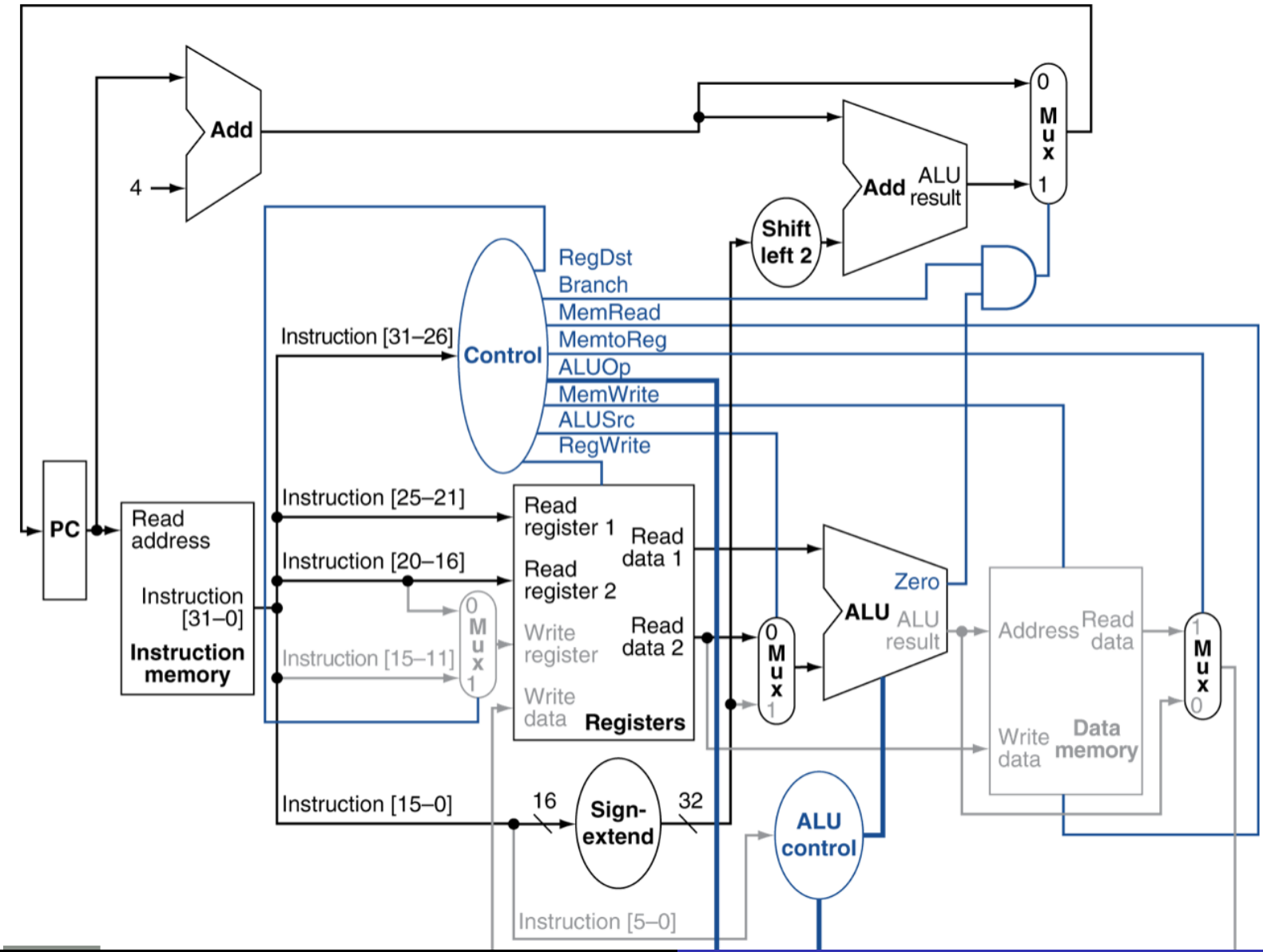


O controle da ALU

Caminho com Controle - Load

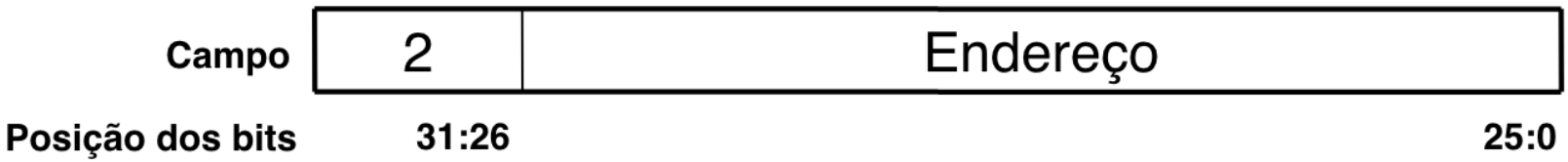


Caminho com Controle - Beq

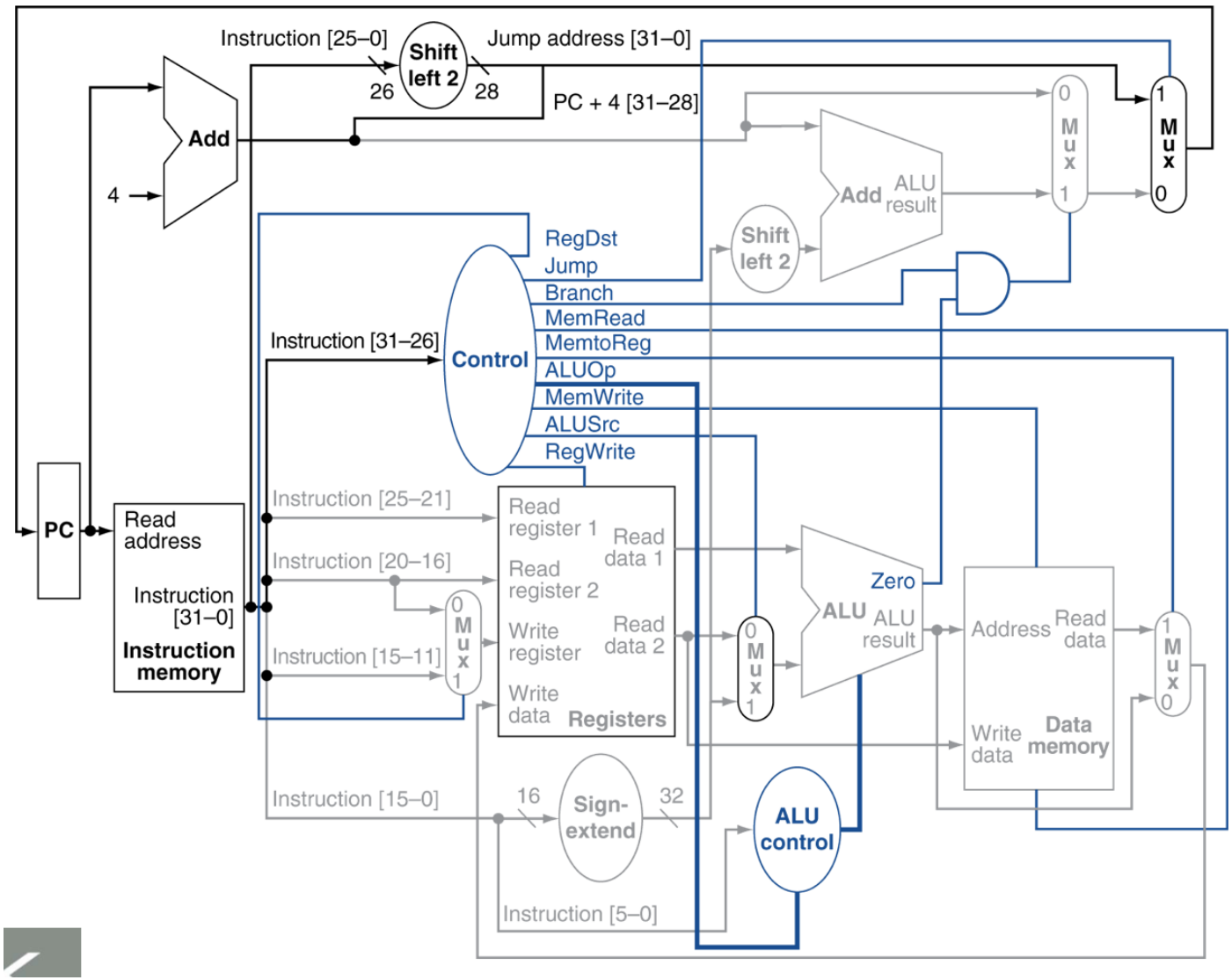


O controle da ALU

- Implementando Jumps
- Endereço de salto é a concatenação de:
  - 2 bits menos significativos, sempre 00
  - 26 bits seguintes vem da instrução
  - 4 bits mais significativos vem de PC+4



Caminho com Controle - Com Jump



Próxima aula

Processador - Caminho de Dados e de Controle - Parte II