

# Unidade de processamento

## Organização básica

João Canas Ferreira

Novembro de 2018



# Tópicos

- 1 Introdução
- 2 Circuitos de tratamento de dados (*datapath*)
- 3 Unidade de controlo

Contém figuras de “Computer Organization and Design – ARM version”, D. Patterson & J. Hennessey, 5ª. ed., MKP

1 Introdução

2 Circuitos de tratamento de dados (*datapath*)

3 Unidade de controlo

# Organização geral

- A eficiência de execução de um dado programa depende do **número de instruções executadas**, do **número médio de ciclos por instrução** (CPI) e do **período do relógio**.
- A **organização interna** determina o período de relógio e o CPI.
- A organização do processador depende do conjunto de instruções. Existem duas grandes abordagens:
  - RISC = Reduced Instruction Set Computer:  
Instruções simples, que facilitam o projeto do CPU, beneficiando CPI e período de relógio em detrimento do nº de instruções: MIPS, Alpha, Sparc.
  - CISC = Complex Instruction Set Computer:  
instruções mais poderosas, com implementação mais complexa: Intel IA-32.
- O processador tem dois módulos principais:
  - 1 unidade de tratamento de dados (*datapath*)
  - 2 unidade de controlo: configura a unidade de tratamento de dados segundo a instrução em execução.

# Especificação do modelo de programação

- Pequeno subconjunto de instruções ARMv8 64-bit (AArch64)

- Banco de registos: 32 registos de 64 bits (X0, X1).

O registo X31 tem sempre 0 (zero). [Nome alternativo: XZR]

- Instruções lógico-aritméticas:

- ADD      X1, X2, X3                       $X0 \leftarrow X1 + X2$

- SUB      X1, X2, X3                       $X0 \leftarrow X1 - X2$

- AND      X1, X2, X3                       $X0 \leftarrow X1 \& X2$

- ORR      X1, X2, X3                       $X0 \leftarrow X1 | X2$


- Instruções de acesso a memória (constante de 9 bits com sinal):

-  ■ LDUR      X0, [X2, #24]                       $X0 \leftarrow \text{Mem}[X2+24]$

-  ■ STUR      X0, [X2, #24]                       $\text{Mem}[X2+24] \leftarrow X0$

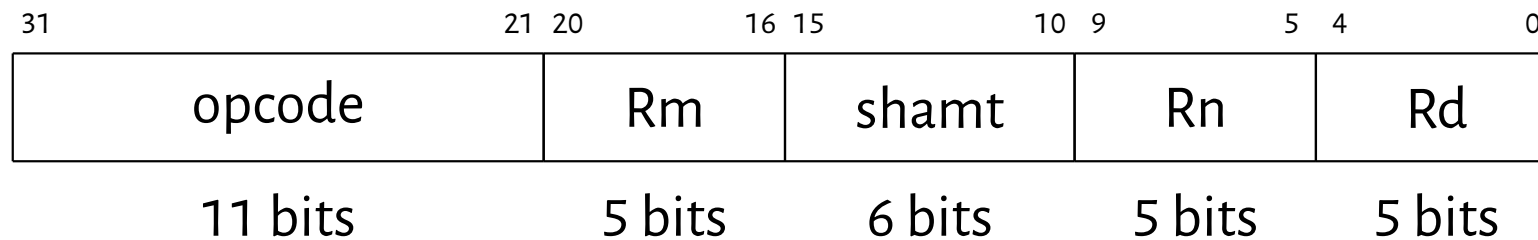
- Instruções de salto:

- B          Label                               $\text{PC} \leftarrow \text{Label}$

-  ■ CBZ      X0, Label                      se X0=0 então  $\text{PC} \leftarrow \text{Label}$

# Codificação das instruções suportadas (I)

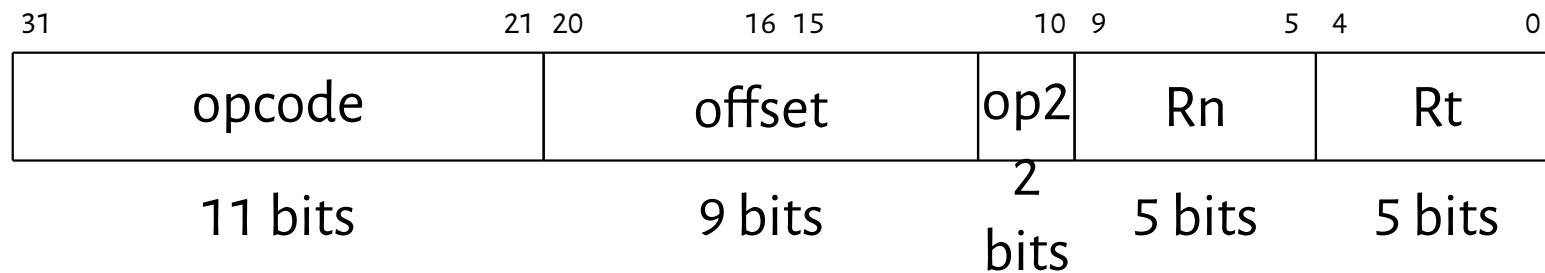
▣ Instruções do tipo R (add, sub, and, orr)



- opcode: operação realizada pela instrução
- 💬 (ADD: 454<sub>H</sub>, SUB: 658<sub>H</sub>, AND: 450<sub>H</sub>, ORR: 550<sub>H</sub>)
- Rm: número do registo com o segundo operando (0–31)
- shamt: (*shift amount*) número de posições a deslocar (para esta implementação é sempre 000000)
- Rn: número do registo com o primeiro operando (0–31)
- Rd: número do registo de destino (0–31)

# Codificação das instruções suportadas (II)

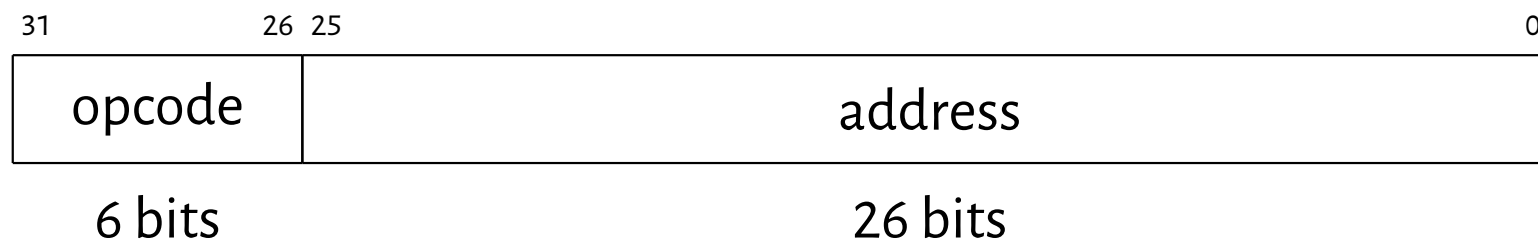
## ▣ Instruções do tipo D (ldur, stur)



- opcode: operação realizada pela instrução (LDUR: 7C2<sub>H</sub>, STUR: 7C0<sub>H</sub>)
- offset: (em cpl/2) endereço = Rn + offset × 4
- op2: expande *opcode*; é 0 nesta implementação
- Rn: número do registo base (0–31)
- Rd: número do registo de destino (0–31)

# Codificação das instruções suportadas (III)

▢ Instruções do tipo B (b): salto incondicional

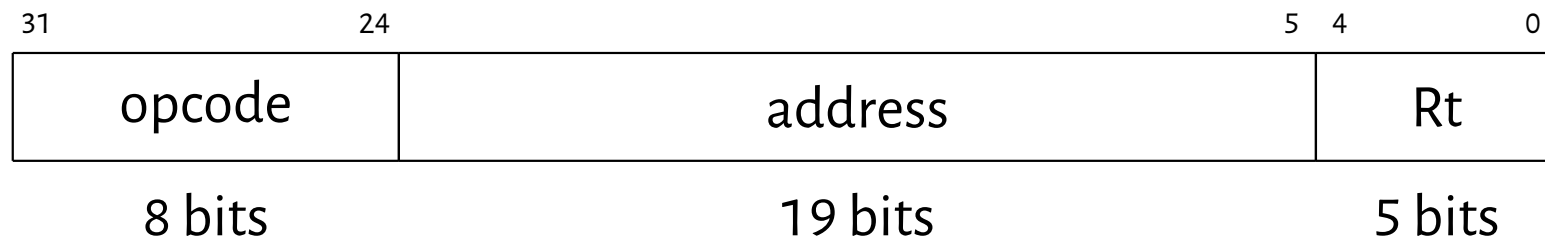


- opcode: operação realizada pela instrução (000101<sub>2</sub>)
  - address: especificação do endereço (relativo ao PC) onde ir buscar a próxima instrução
- $$PC \leftarrow PC + \text{address} \times 4$$



# Codificação das instruções suportadas (IV)

▣ Instruções do tipo CB (cbz): salto condicional



- opcode: operação realizada pela instrução ( $10110110_2$ )
- address: especificação do endereço (relativo ao PC) onde ir buscar a próxima instrução  
 $PC \leftarrow PC + \text{address} \times 4$
- número do registo a usar na comparação (com zero)

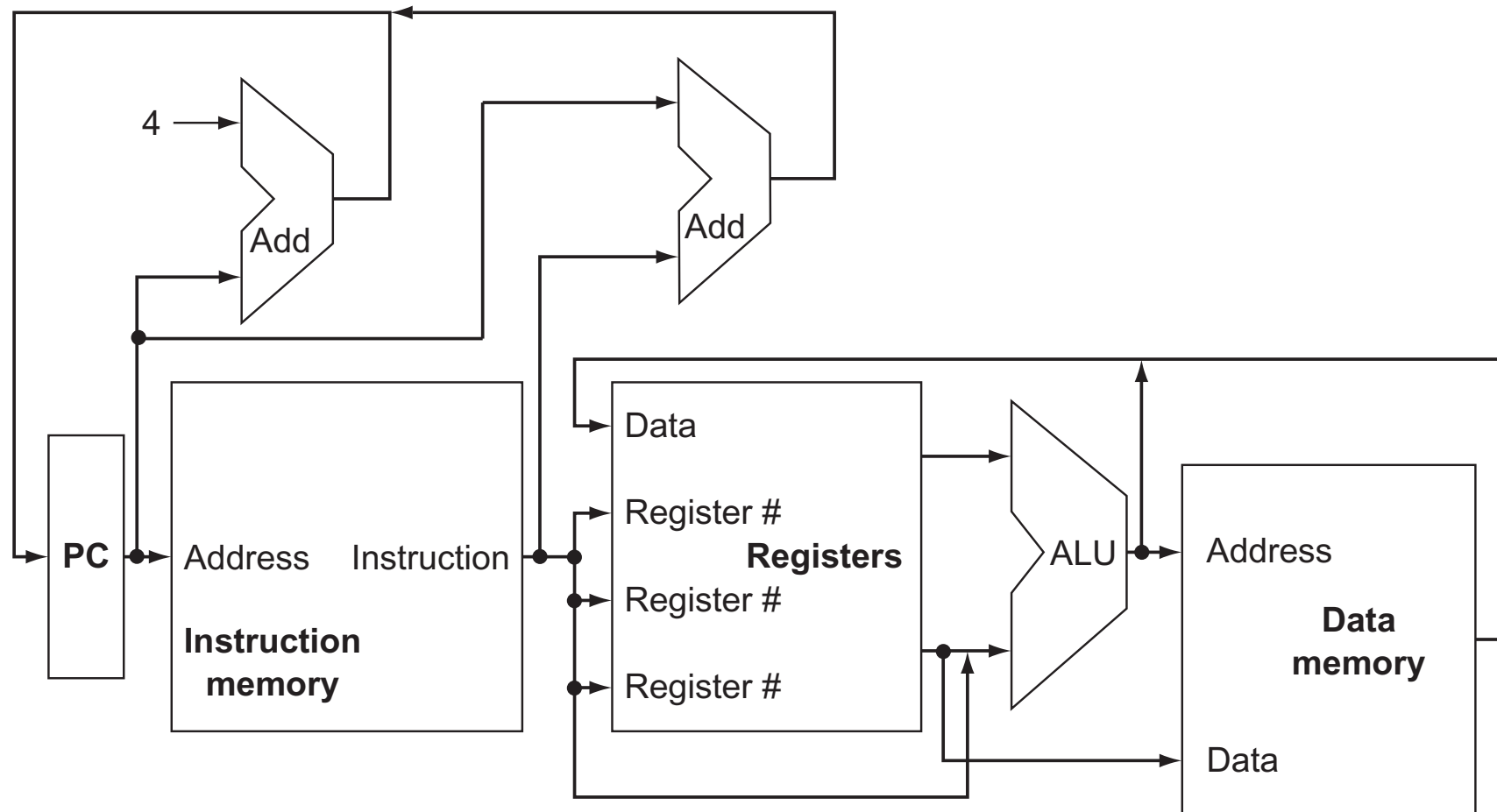
# Aspetos comuns a todas as instruções

▣ Para definir a organização do CPU, é conveniente determinar quais aspetos são comuns a (quase) todas as instruções?

- Os dois primeiros passos do tratamento de todas as instruções são idênticos:
  - 1 Enviar o conteúdo do contador de programa (PC) para a memória de instruções e obter a instrução.
  - 2 Ler *um ou dois registos* (usando os campos da instrução para os seleccionar).  
A instrução LDUR apenas necessita de ler um registo (o valor do 2º registo pode ser ignorado).
- As ações subsequentes dependem da instrução, mas são semelhantes para instruções da mesma classe.
- Mesmo instruções de classes diferentes têm semelhanças. Por exemplo, todas as instruções (exceto salto incondicional) usam a ALU:
  - lógico-aritméticas: ALU efectua a operação;
  - *load/store*: ALU é usada para cálculo do endereço efetivo;
  - salto condicional: ALU é usada para efetuar a comparação.

# Componentes e fluxo de informação do CPU básico

Com base nos aspetos comuns, vamos considerar a seguinte organização:



Fonte: [COD5-ARM]

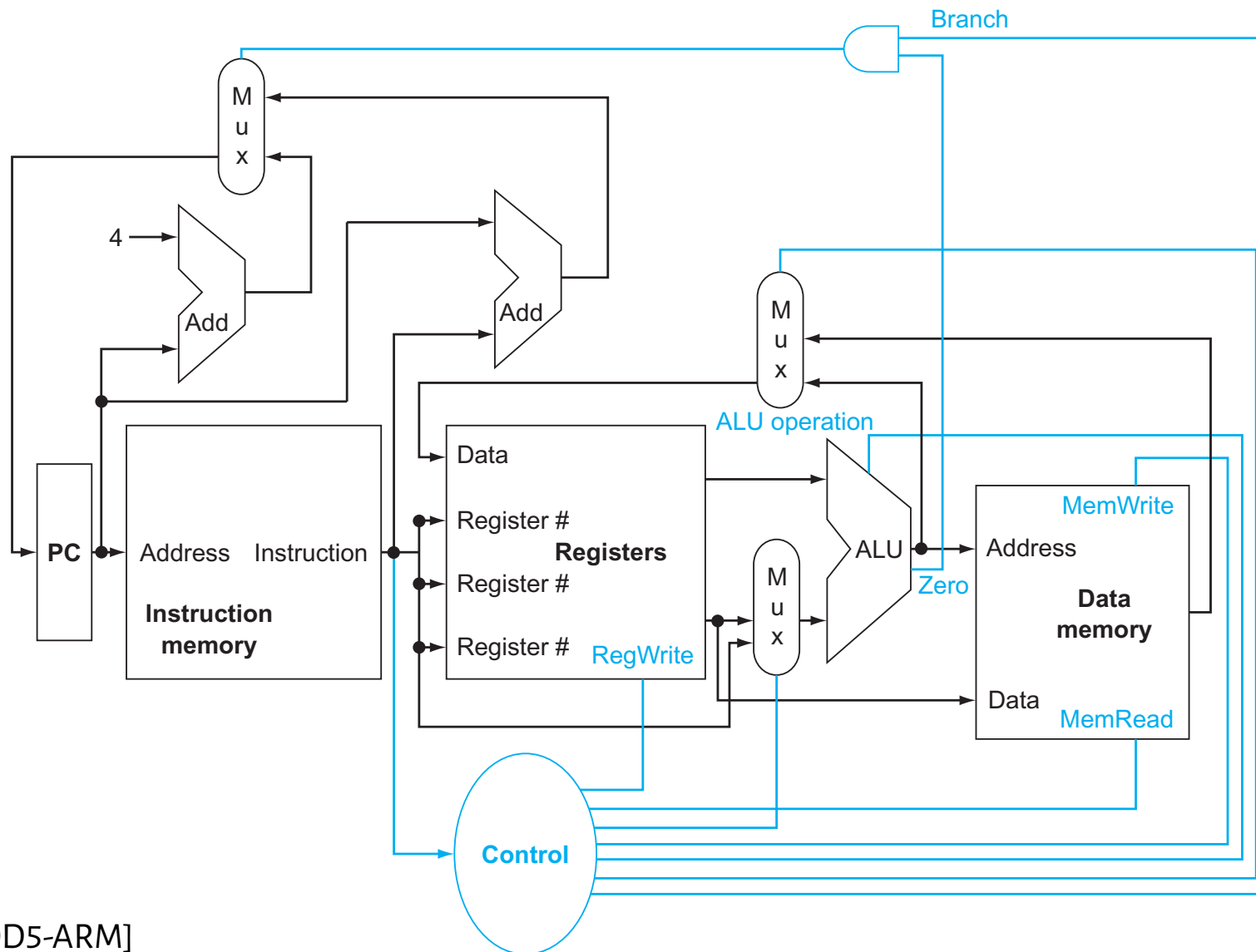
Diagrama apresenta apenas o fluxo de informação; não é (ainda) o diagrama de blocos de um circuito.

# Lista de componentes a usar

▢ Que componentes lógicos são necessários?

- Memória de instruções
- Memória de dados
  - esta separação corresponde a um modelo simplificado da hierarquia de memória
- Contador de programa (PC)
- Banco de registos com três portos:
  - dois portos de leitura;
  - um porto de escrita.
- Unidade lógico-aritmética (ALU)
- Vários somadores
- Multiplexadores
  - para todas as entradas que têm diversas fontes
- Unidade de controlo
  - para comandar os multiplexadores (definir o valor dos sinais de controlo)

# Diagrama de blocos do CPU básico (quase completo)



Fonte: [COD5-ARM]

A azul: unidade de controlo e respetivas ligações.

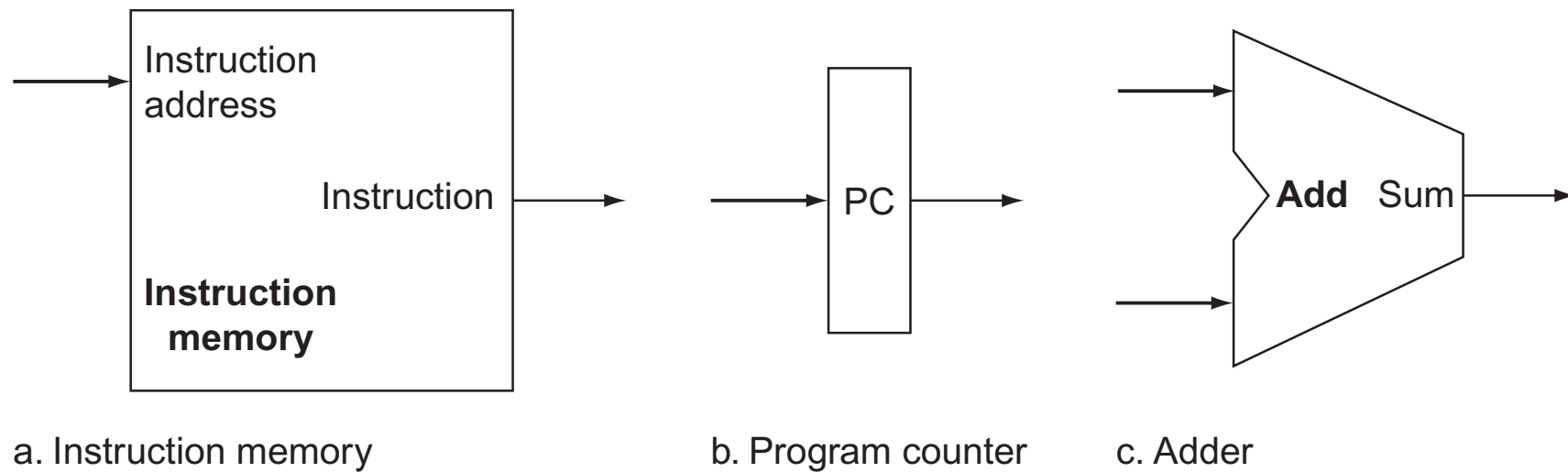
1 Introdução

2 Circuitos de tratamento de dados (*datapath*)

3 Unidade de controlo

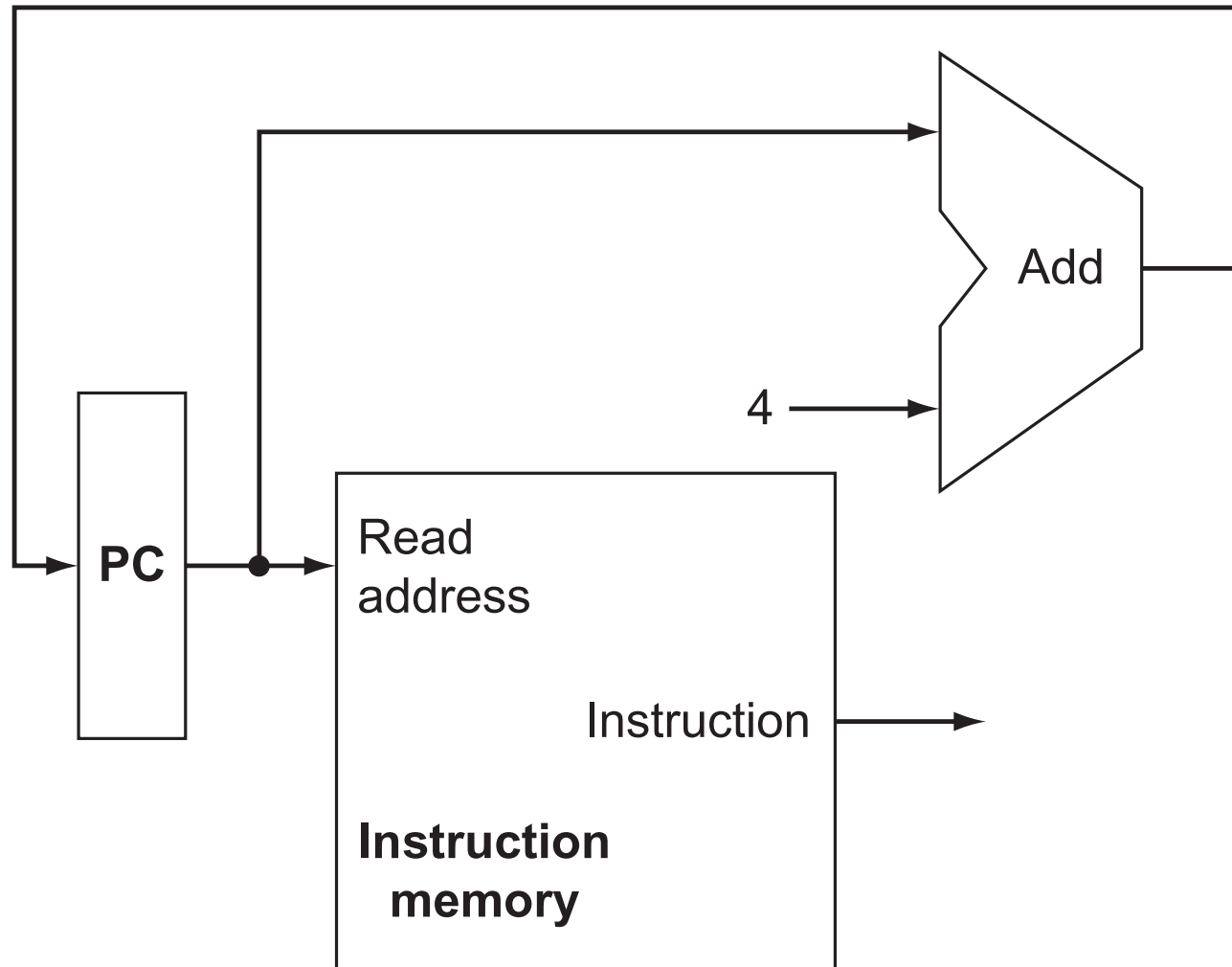
# Componentes para obtenção de instruções

➡ Componentes lógicos necessários para a tarefa de obtenção de instruções (*instruction fetch*):



Fonte: [COD5-ARM]

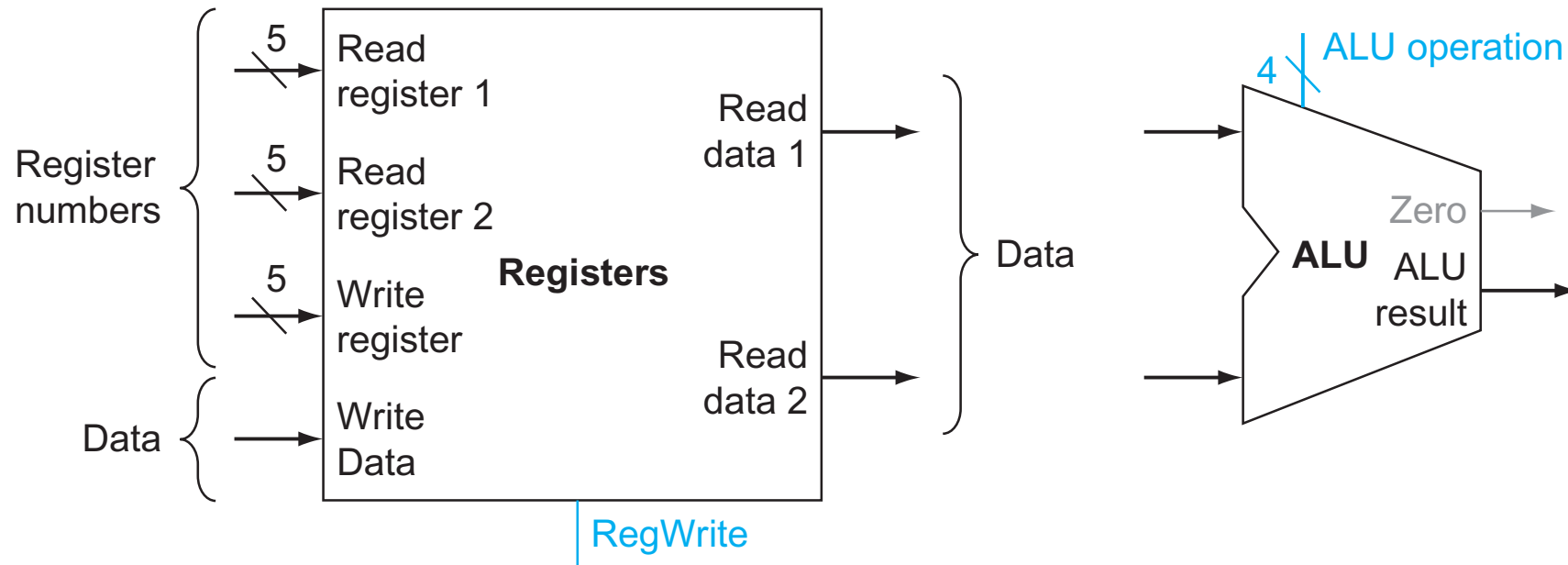
# Circuito para obtenção de instruções



Fonte: [COD5-ARM]



# Componentes para executar instruções do tipo R

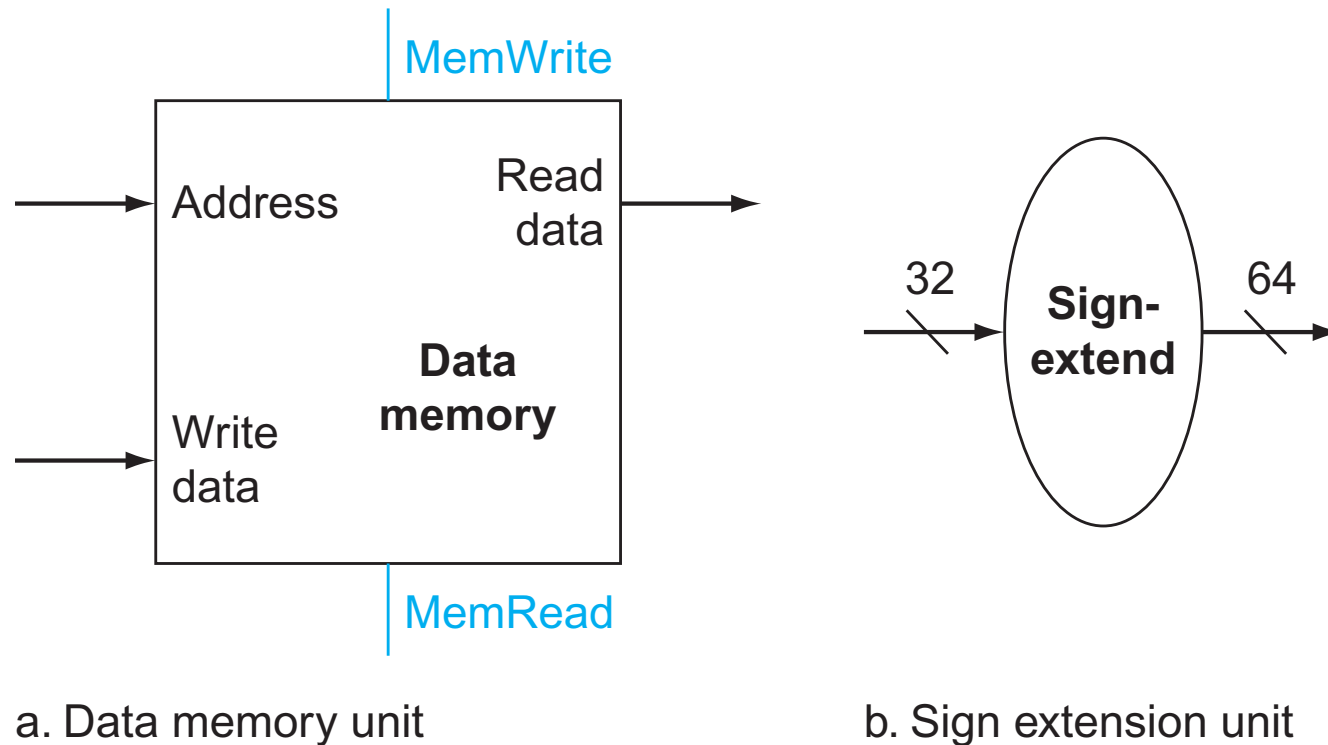


a. Registers  
Fonte: [COD5-ARM]

b. ALU

Sinais de controlo da ALU	Função
0000	AND
0001	OR
0010	add
0110	subtract
0111	$(A < B) ? 1 : 0$
1100	NOR

# Componentes para acessos a memória de dados

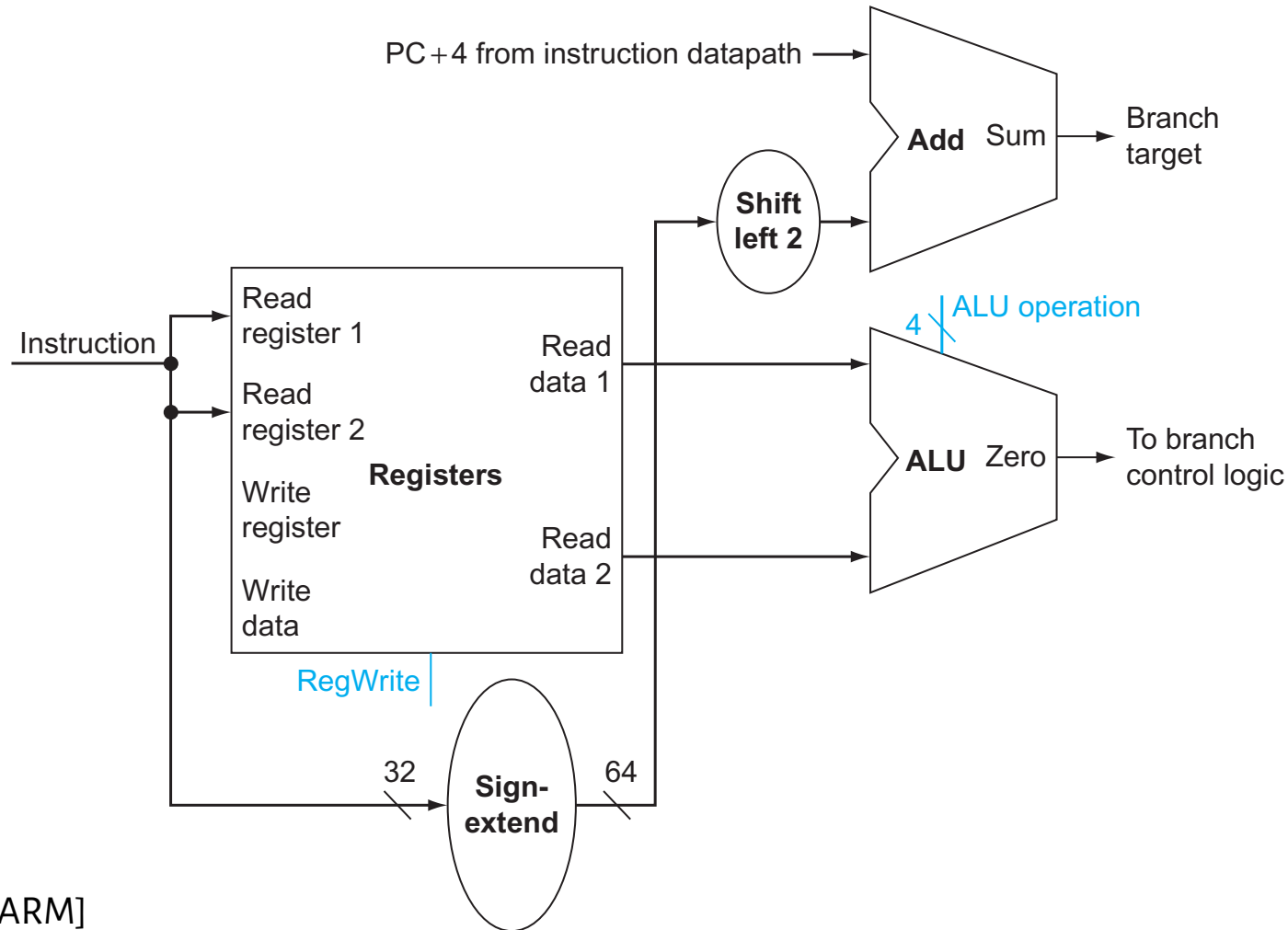


Fonte: [COD5-ARM]

- **MemRead**: sinal de habilitação de leitura
  - ➡ Necessário para evitar a leitura de endereços “ilegais”, que podem surgir na entrada Address durante a execução de outras instruções que não acessem a memória.
- **MemWrite**: sinal de habilitação de escrita

# Componentes para avaliação da condição de salto

(Nota: apenas para a instrução CBZ.)

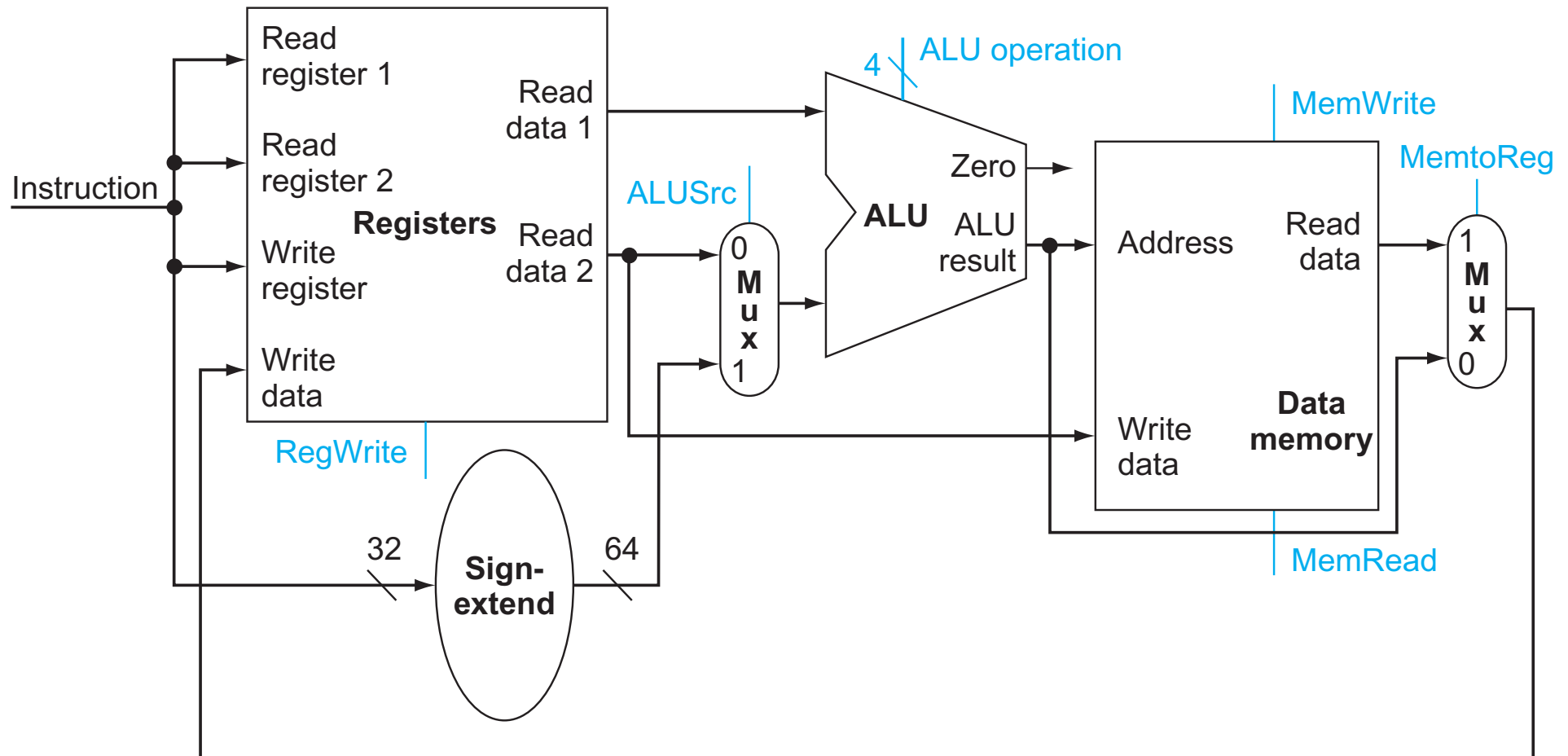


Fonte: [COD5-ARM]

➡ A operação da ALU é “subtração” (ALU operation = 0110).

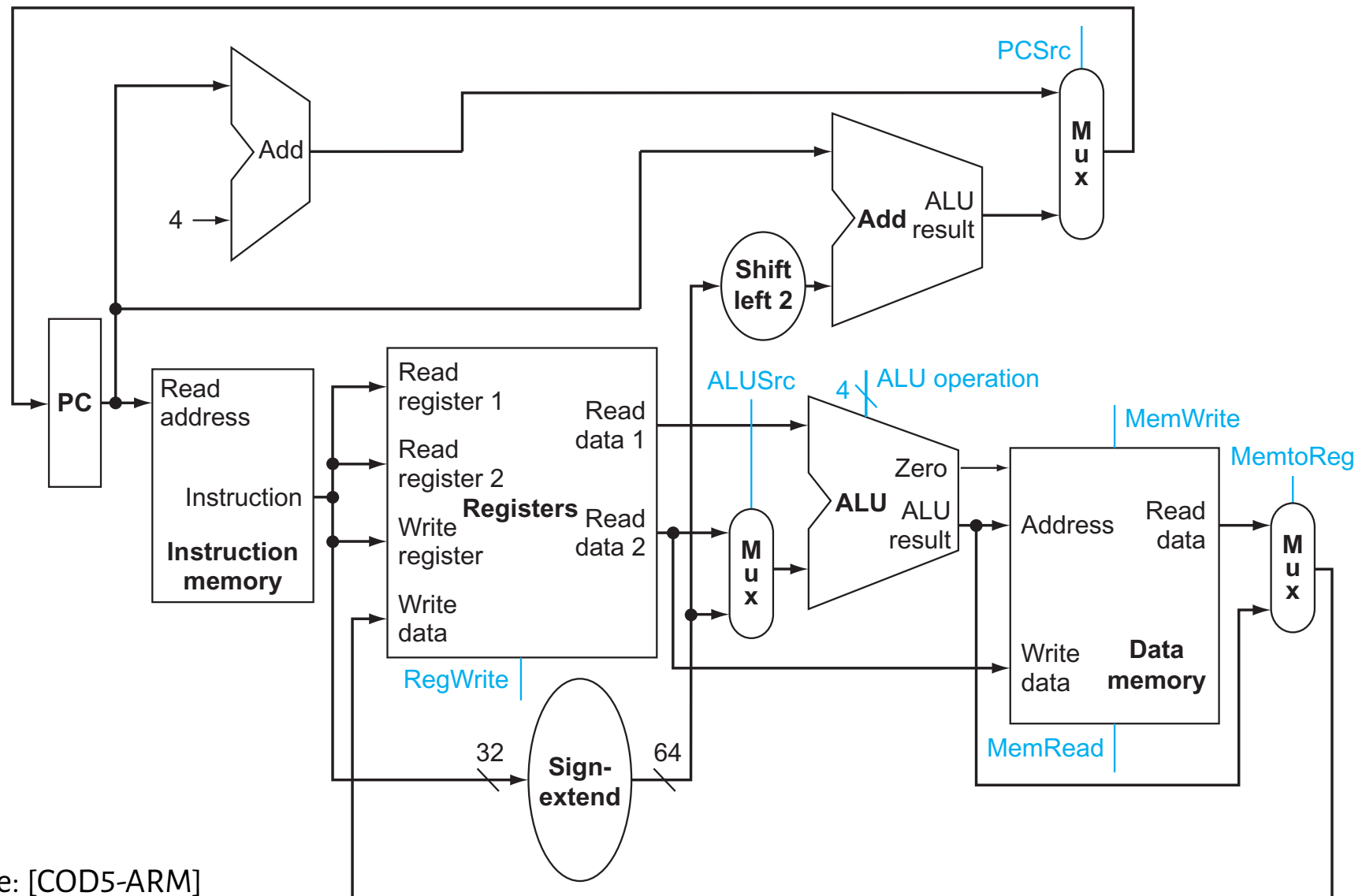
# Instruções tipo R e acessos a memória

- Combinar os circuitos para instruções tipo R e acessos a memória



Fonte: [COD5-ARM]

# Caminho de dados (quase) completo



Fonte: [COD5-ARM]

Falta suporte para saltos incondicionais.