Unidade de processamento

Organização básica

João Canas Ferreira

Novembro de 2018

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

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

Organização geral

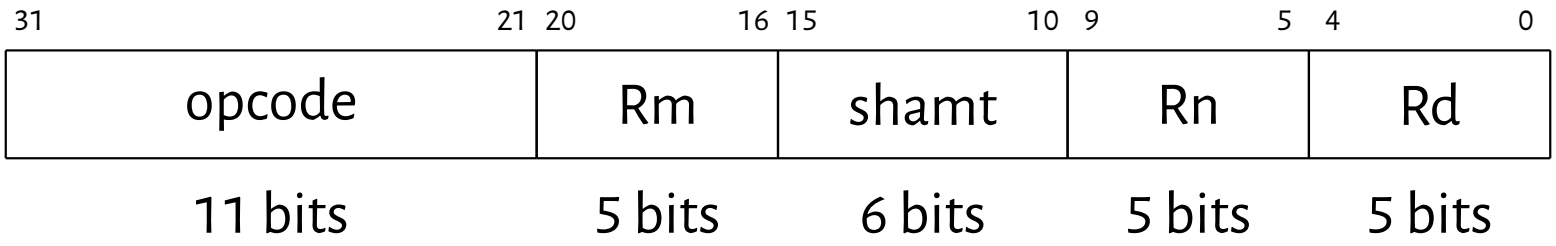
- A eficiência de execução de um dado programa depende do **número de instruções executadas**, do **número médio de ciclos por instrução** (CPI) e do **período do relógio**.
- A **organização interna** determina o período de relógio e o CPI.
- A organização do processador depende do conjunto de instruções. 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` ← `X1 + X2`
  - `SUB X1, X2, X3` — `X0` ← `X1 - X2`
  - `AND X1, X2, X3` — `X0` ← `X1 & X2`
  - `ORR X1, X2, X3` — `X0` ← `X1 | X2`
- Instruções de acesso a memória (constante de 9 bits com sinal) para *doublewords*(8 bytes):
  - `LDUR X0, [X2,#24]` — `X0` ← `Mem[X2+24]`
  - `STUR X0, [X2,#24]` — `Mem[X2+24]` ← `X0`
- Instruções de salto:
  - `B Label` — PC ← Label
  - `CBZ X0, Label` — se `X0=0` então PC ← Label

Codificação das instruções suportadas (I)

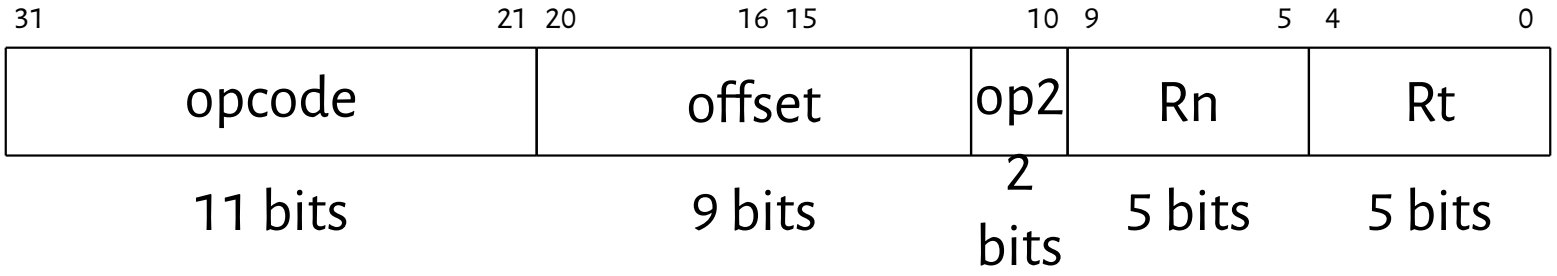
- Instruções do tipo R (add, sub, and, orr)



- opcode: operação realizada pela instrução
  (ADD: 454_H, SUB: 658_H, AND: 450_H, ORR: 550_H)
- 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)

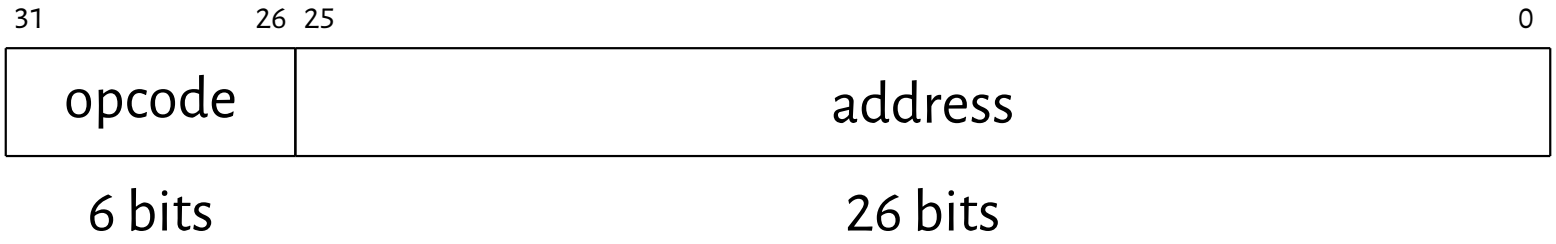
- Instruções do tipo D (ldur, stur)



- opcode: operação realizada pela instrução (LDUR: $7C2_H$, STUR: $7C0_H$)
- offset: (em cpl/2) endereço = Rn + offset
- 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

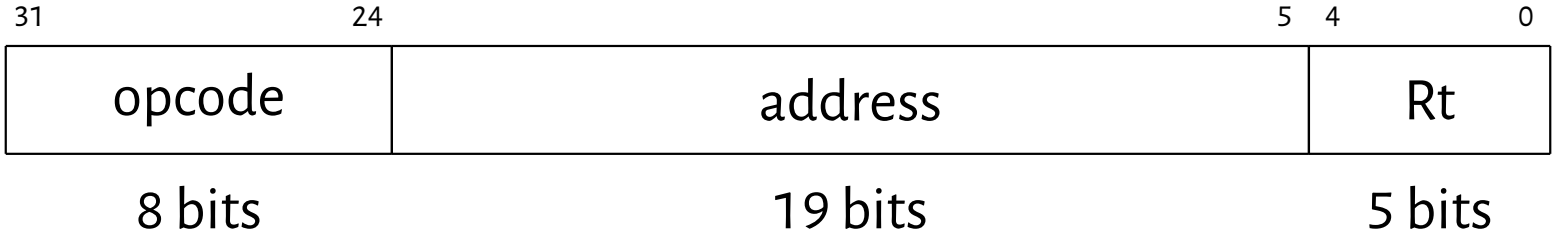


  - opcode: operação realizada pela instrução (000101_2)
  - address: especificação do endereço (relativo ao PC) onde ir buscar a próxima instrução

    PC⟵PC + address ×4

Codificação das instruções suportadas (IV)

- Instruções do tipo CB (cbz): salto condicional



- opcode: operação realizada pela instrução (10110100_2)
- address: especificação do endereço (relativo ao PC) onde ir buscar a próxima instrução

  PC←PC + address ×4
- número do registo a usar na comparação (com zero)

Aspetos comuns a todas as instruções

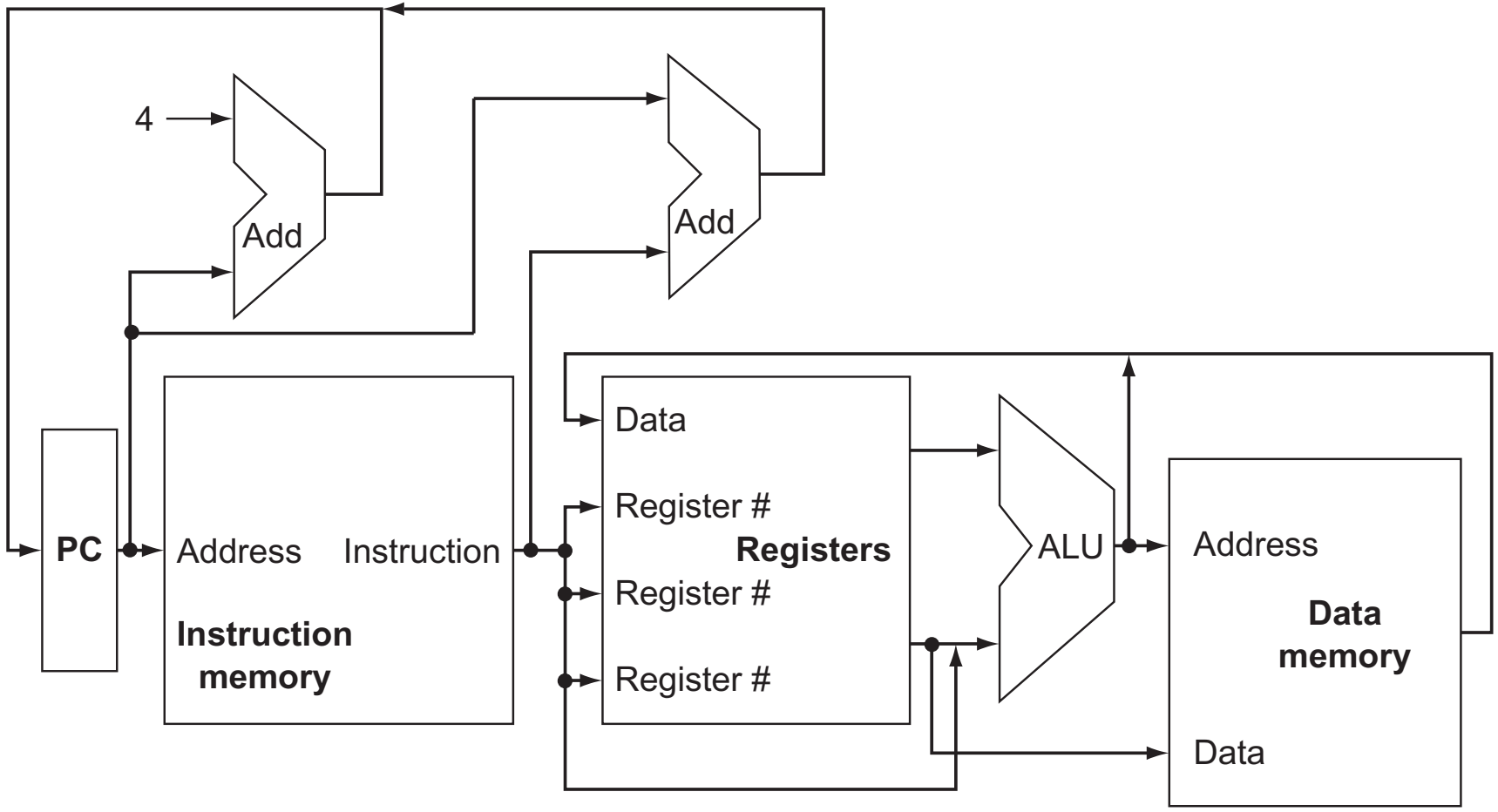
(Nota: Para já, não considerar a instrução de salto incondicional.)

➠ 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 selecionar). 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:



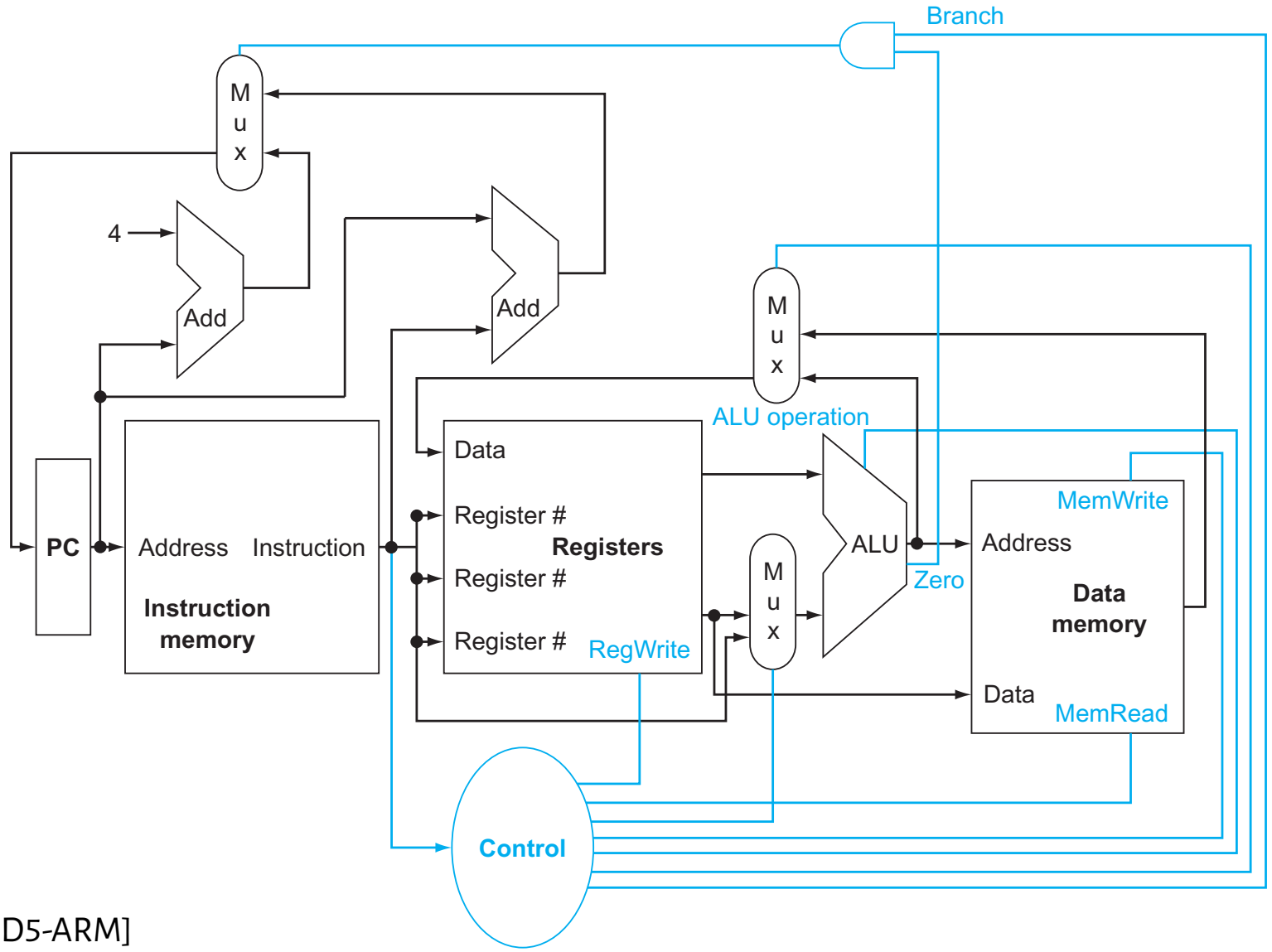
Fonte: [COD5-ARM]

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.

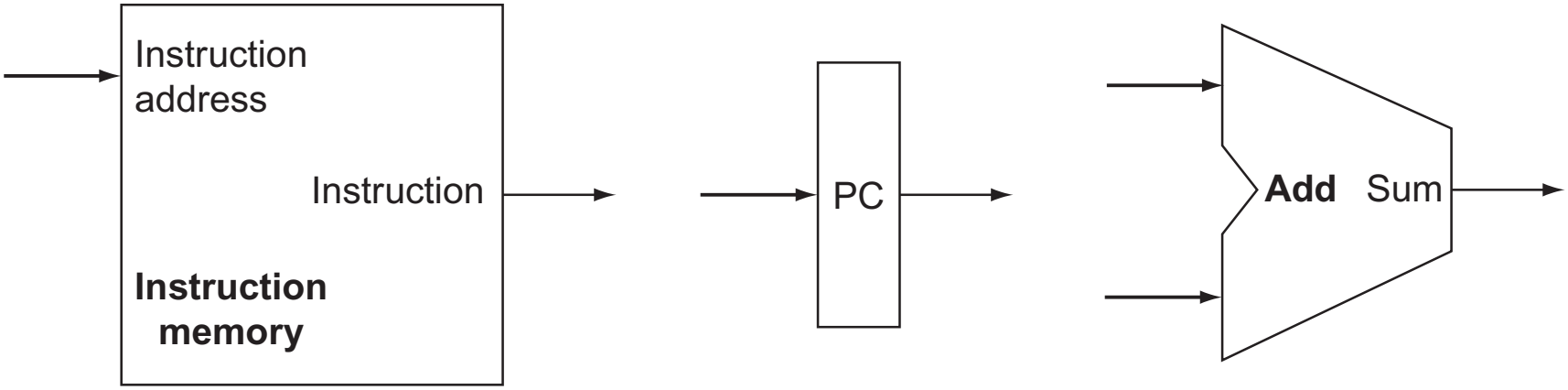
1 Introdução

2 Circuitos de tratamento de dados *(datapath)*

3 Unidade de controlo

Componentes para obtenção de instruções

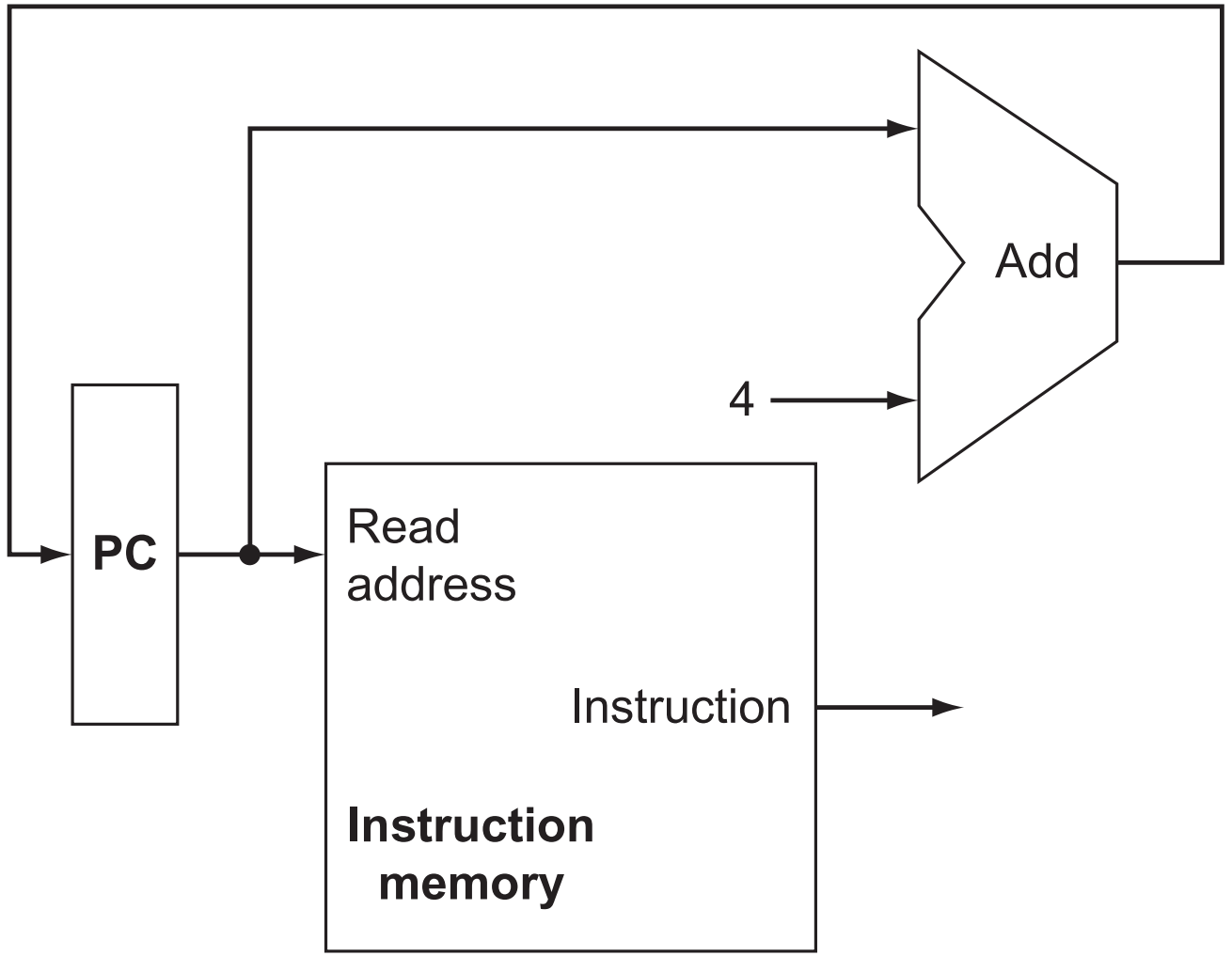
➠ Componentes lógicos necessários para a tarefa de obtenção de instruções *(instruction fetch)*:



a. Instruction memory b. Program counter c. Adder

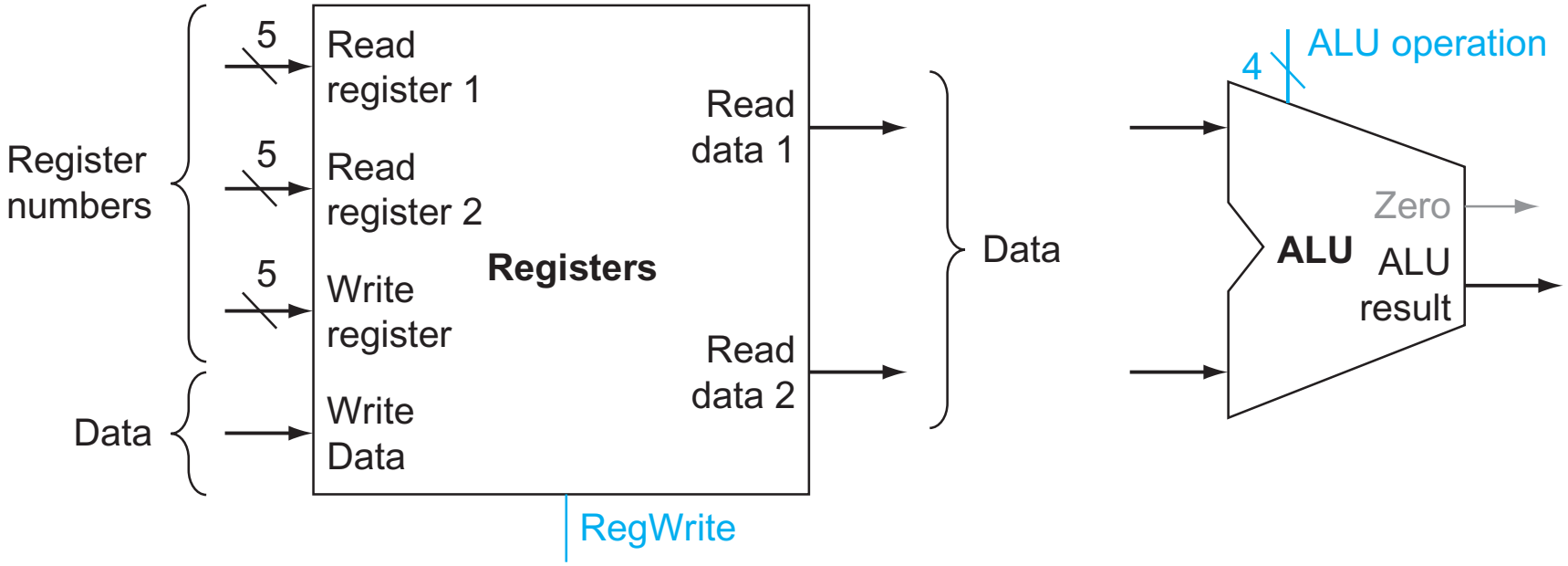
Fonte: [COD5-ARM]

Circuito para obtenção de instruções



Fonte: [COD5-ARM]

Componentes para executar instruções do tipo R

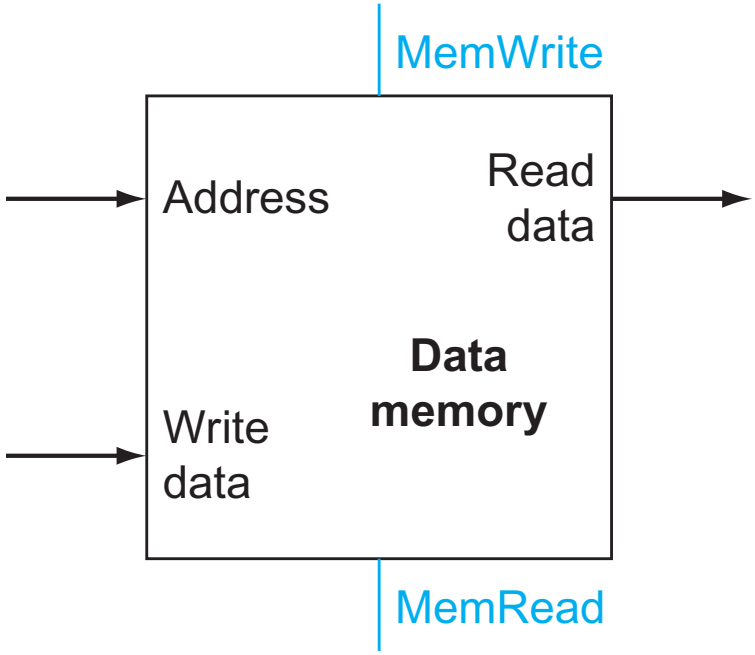


a. Registers b. ALU

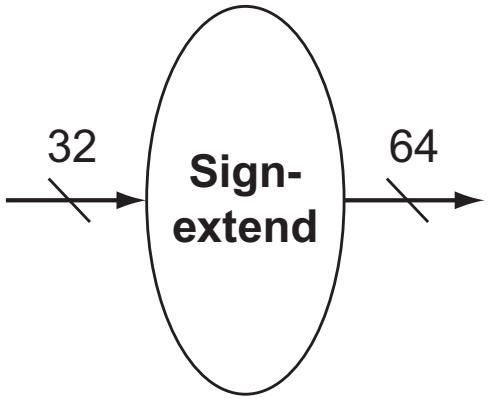
Fonte: [COD5-ARM]

Sinais de controlo da ALU	Função
0000	AND
0001	OR
0010	adição
0110	subtração
0111	transferir entrada B
1100	NOR

Componentes para acessos a memória de dados



a. Data memory unit



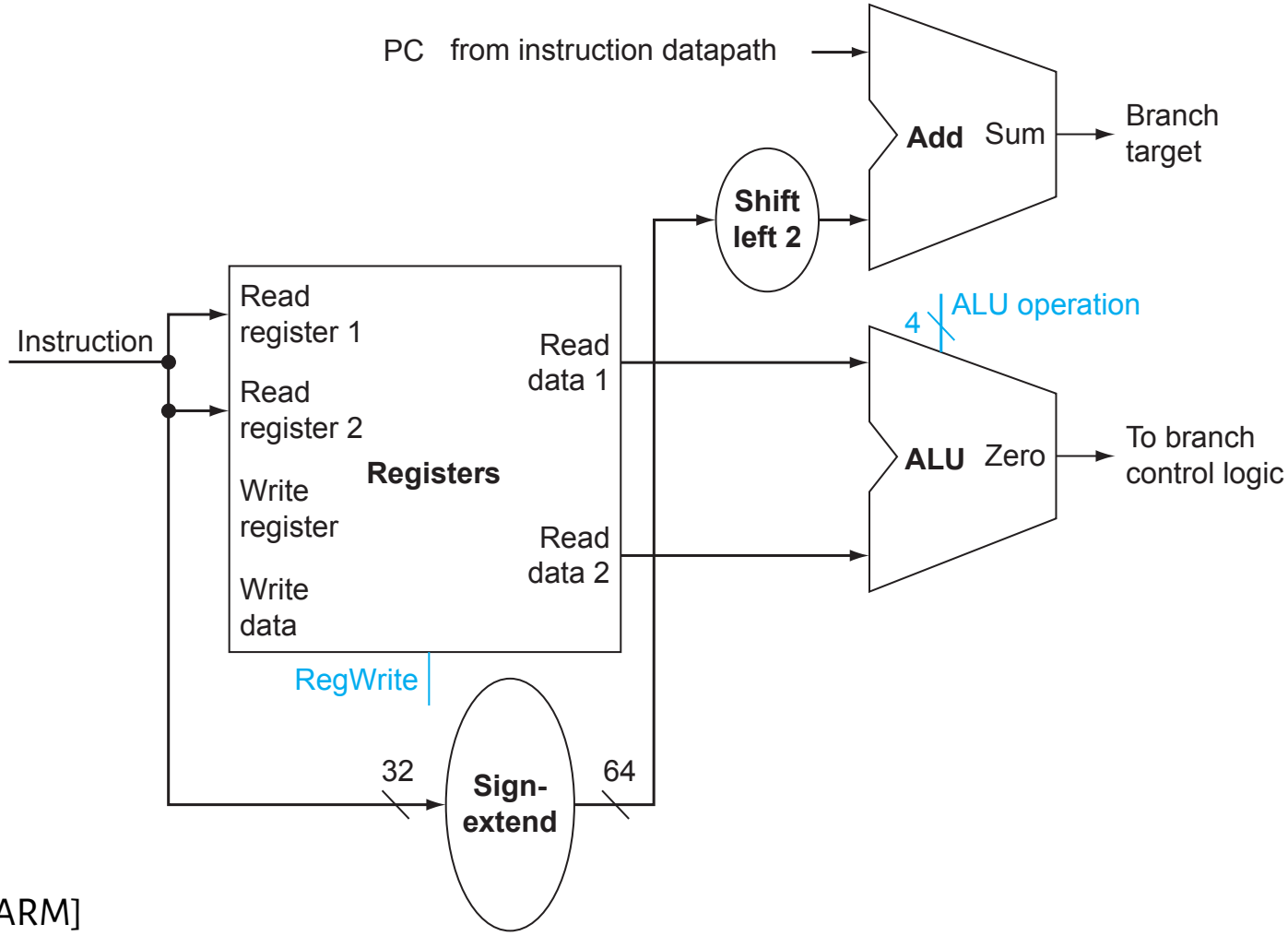
b. Sign extension unit

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 acedam a memória.
- `MemWrite`: sinal de habilitação de escrita
- Extensão de sinal: extrair campo de 9 ou 19 bits e expandir para 64 (usar bit 26 da instrução: 0 para transferências de dados, 1 para CBZ)

Componentes para avaliação da condição de salto

(Nota: apenas para a instrução CBZ.)

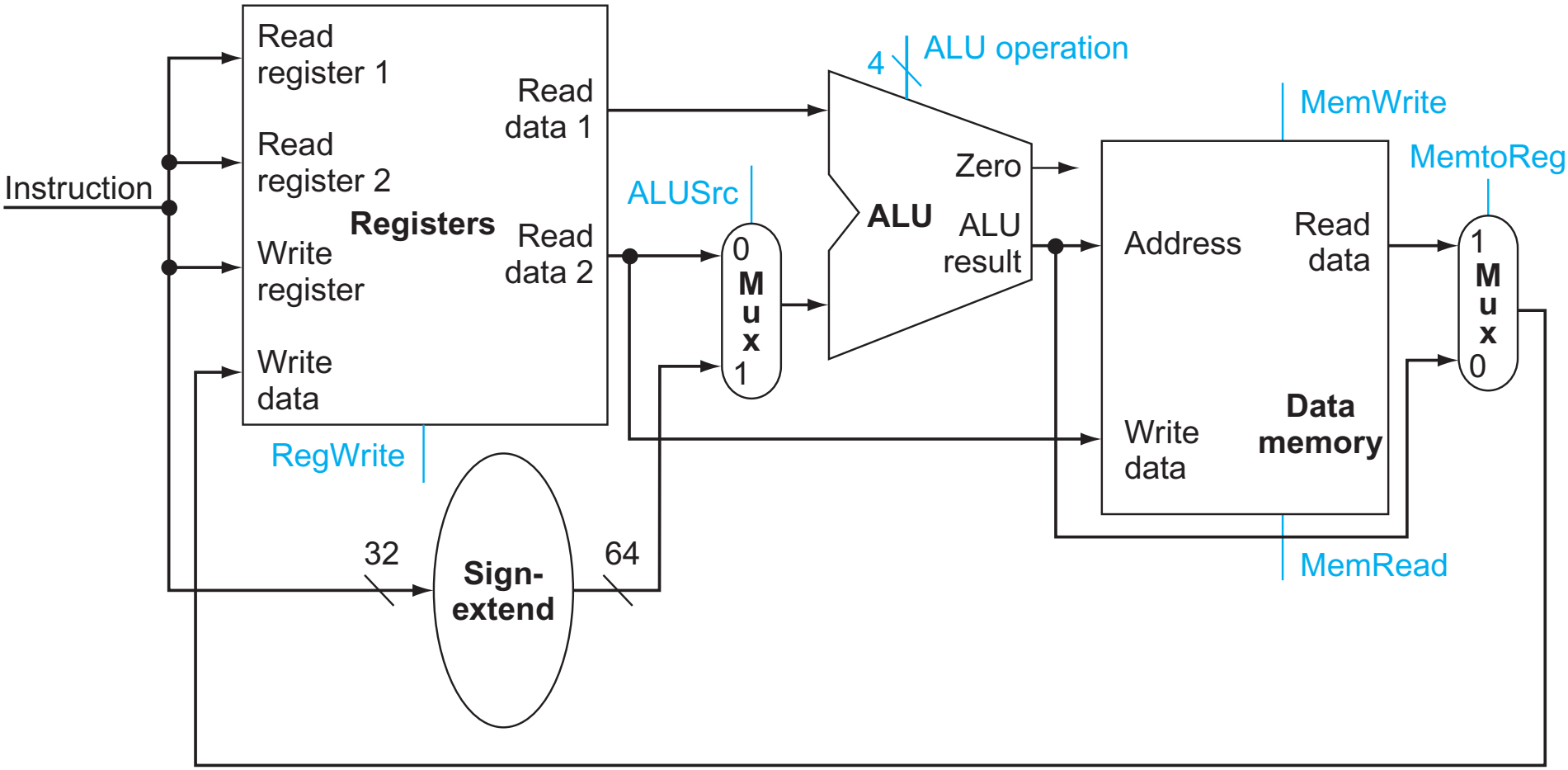


Fonte: [COD5-ARM]

➠ A operação da ALU é "transferir entrada B" (`ALU operation = 011`).

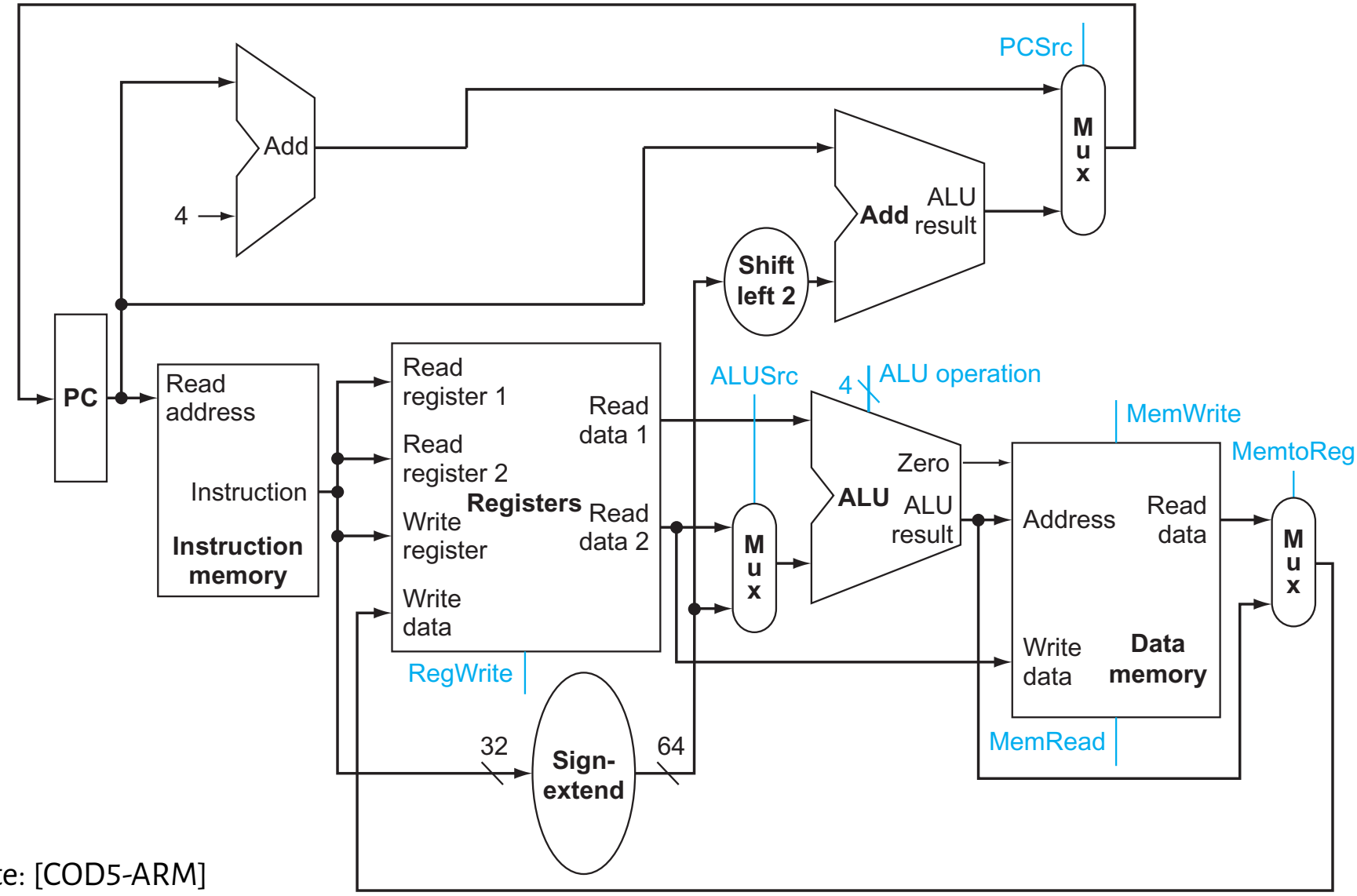
Instruções tipo R e acessos a memória

- 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.

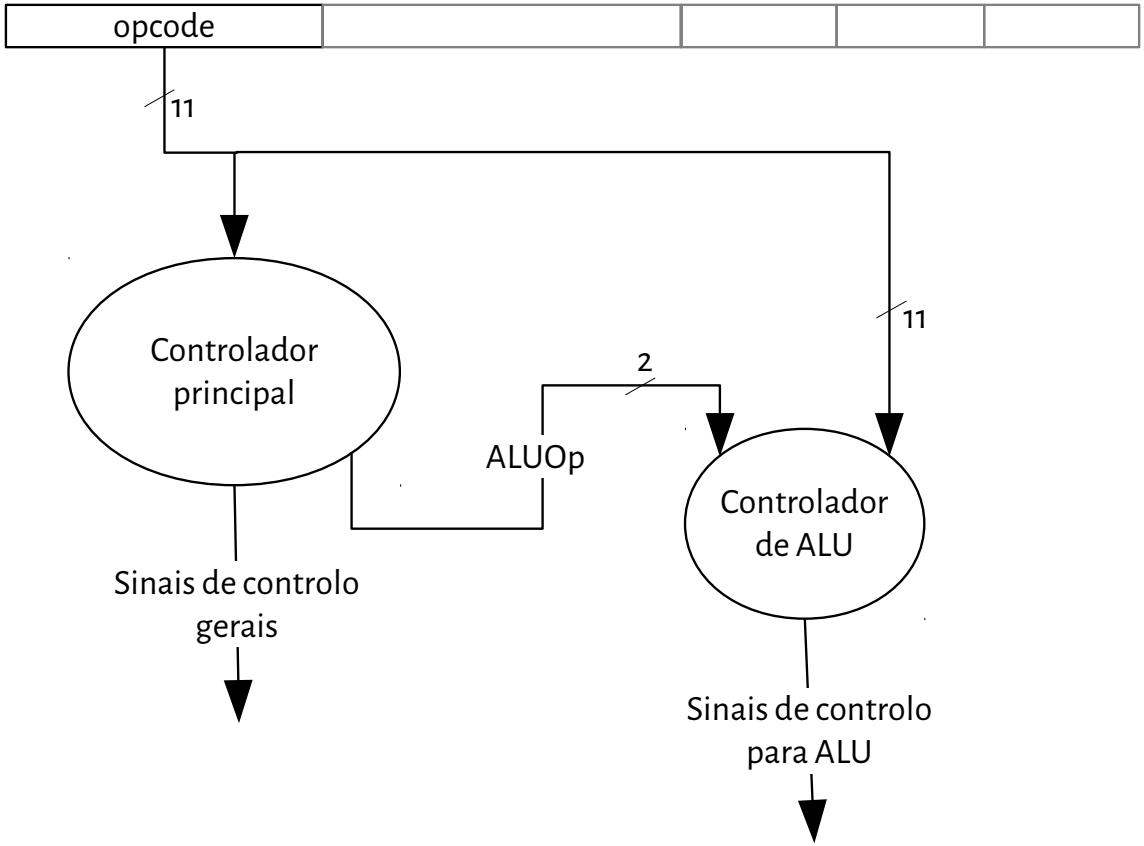
1 Introdução

2 Circuitos de tratamento de dados *(datapath)*

3 Unidade de controlo

Controlo da ALU

➠ Estratégia: Separar a unidade de controlo em duas.



- ALU trabalha em três contextos diferentes.
  1. instruções lógico-aritméticas: `ALUOp[1:0]=10`
  2. cálculo de endereços: `ALUOp[1:0]=00`
  3. comparação: `ALUOp[1:0]=01`

Especificação do controlador de ALU

Entradas: instrução e ALUOp[1:0] Saídas: CtrlALU[3:0]

instrução	ALUOp	Operação	opcode	Descrição	CtrlALU
LDUR	00	load register	XXXXXXXXXXX	adição	0010
STUR	00	store register	XXXXXXXXXXX	adição	0010
CBZ	01	branch equal 0	XXXXXXXXXX	transferir	0111
R	10	ADD	10001011000	adição	0010
R	10	SUB	11001011000	subtração	0110
R	10	AND	10001010000	e-lógico	0000
R	10	ORR	10101010000	ou-lógico	0001

A coluna **Ctrl ALU** especifica os valores que a entrada de controlo da ALU deve receber para que execute a operação apropriada a cada contexto (instrução).

ALUOp		Opcode field											Operation
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[24]	I[23]	I[22]	I[21]	
0	0	X	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	X	0111
1	X	1	0	0	0	1	0	1	1	0	0	0	0010
1	X	1	1	0	0	1	0	1	1	0	0	0	0110
1	X	1	0	0	0	1	0	1	0	0	0	0	0000
1	X	1	0	1	0	1	0	1	0	0	0	0	0001

Observações gerais sobre a descodificação

Field	opcode	Rm	shamt	Rn	Rd
Bit positions	31:21	20:16	15:10	9:5	4:0

a. R-type instruction

Field	1986 or 1984	address	0	Rn	Rt
Bit positions	31:21	20:12	11:10	9:5	4:0

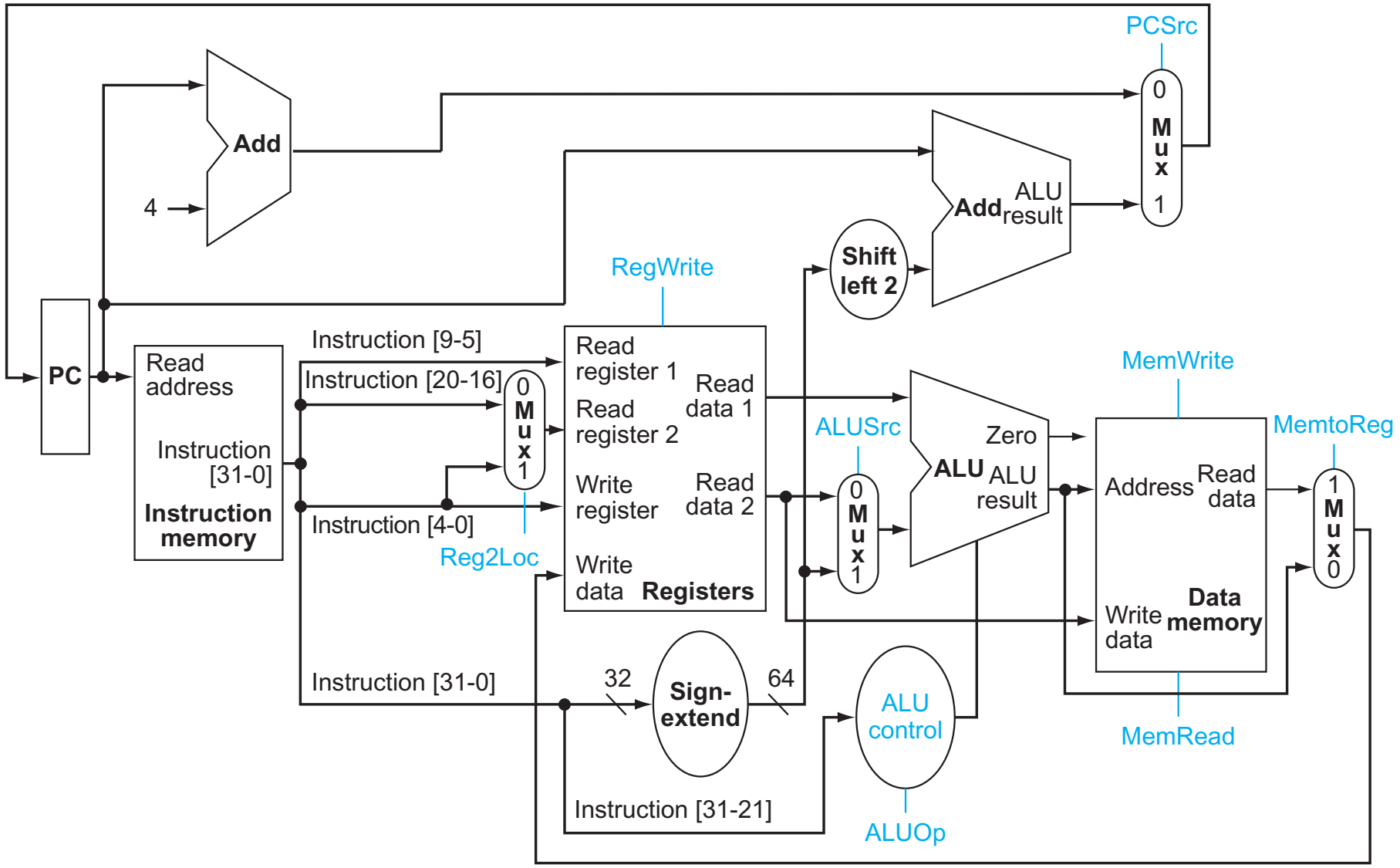
b. Load or store instruction

Field	180	address	Rt
Bit positions	31:24	23:5	4:0

c. Conditional branch instruction

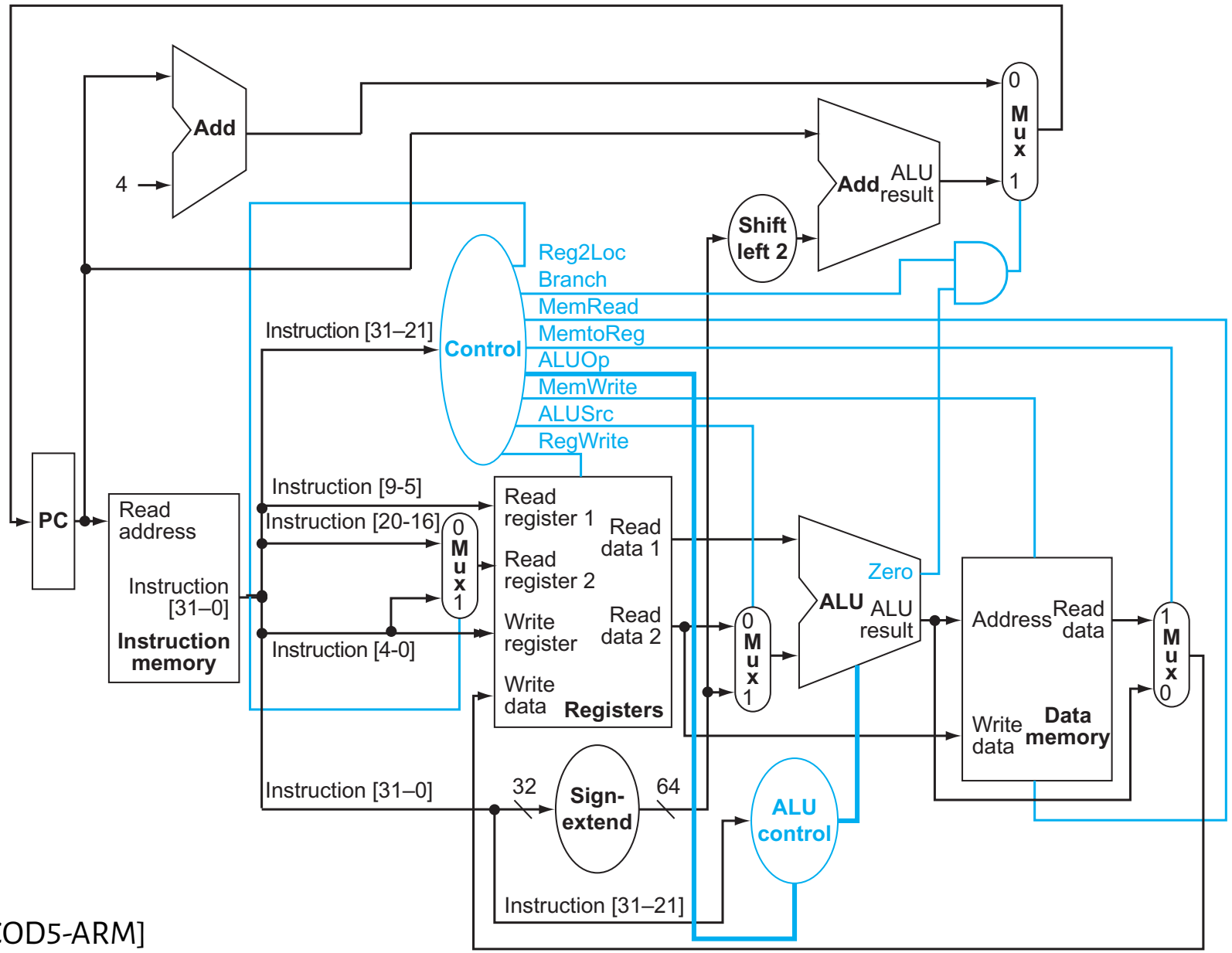
- Código da instrução nas posições 31:21 ou 31:24.
- Registo de destino na posição 4:0 (instruções tipo R e LDUR)
- 1º operando dos cálculos na posições 9:5 (instruções tipo R ou D)
- 2º operando está nas nas posições 20:16 (Rm) ou 4:0 (Rt) [para CBZ e LDUR]
  - Usar um multiplexador para escolher registo a usar como 2º operando.
- 2º operando pode ser um valor imediato de 19 bits (CBZ) ou de 9 bits (LDUR e STUR)

Caminho de dados com multiplexadores e sinais controlo



Fonte: [COD5-ARM]

Detalhes das ligações da unidade de controlo



Fonte: [COD5-ARM]

Especificação dos sinais de controlo

Nome	Se valor = 0	Se valor = 1
Reg2Loc	O número do registo 2 vem do campo 20:16 (Rm)	O número do registo 2 vem do campo 4:0 (Rt)
RegWrite	Nenhum	O registo do campo 4:0 recebe o valor da entrada `Write data`
ALUSrc	2º operando de ALU vem da saída 2 do banco de registos.	2º operando da ALU é constante da instrução
PCSrc	PC assume valor PC+4	PC ←destino do salto
MemRead	Nenhum	Memória fornece valor pedido
MemWRite	Nenhum	Memória guarda valor especificado por entrada `WriteData`
MemtoReg	O valor a escrever no banco de registos provém da ALU	O valor a escrever no banco de registos provém da memória
Branch	Instrução não é CBZ	Instrução é CBZ

- PCSrc = Branch · Zero

Especificação do controlador principal

➠ As tabelas seguintes indicam os valores que os sinais de controlo devem assumir durante o tratamento de cada instrução.

Instrução	Reg2Loc	ALUSrc	MemtoReg	RegWrite
R	0	0	0	1
LDUR	X	1	1	1
STUR	1	1	X	0
CBZ	1	0	X	0

Instrução	MemRead	MemWrite	Branch	ALUop[1]	ALUop[0]
R	0	0	0	1	0
LDUR	1	0	0	0	0
STUR	0	1	0	0	0
CBZ	0	0	1	0	1

➠ Para obter tabelas de verdade é preciso substituir o nome das instruções pelo seu *(opcode)*.

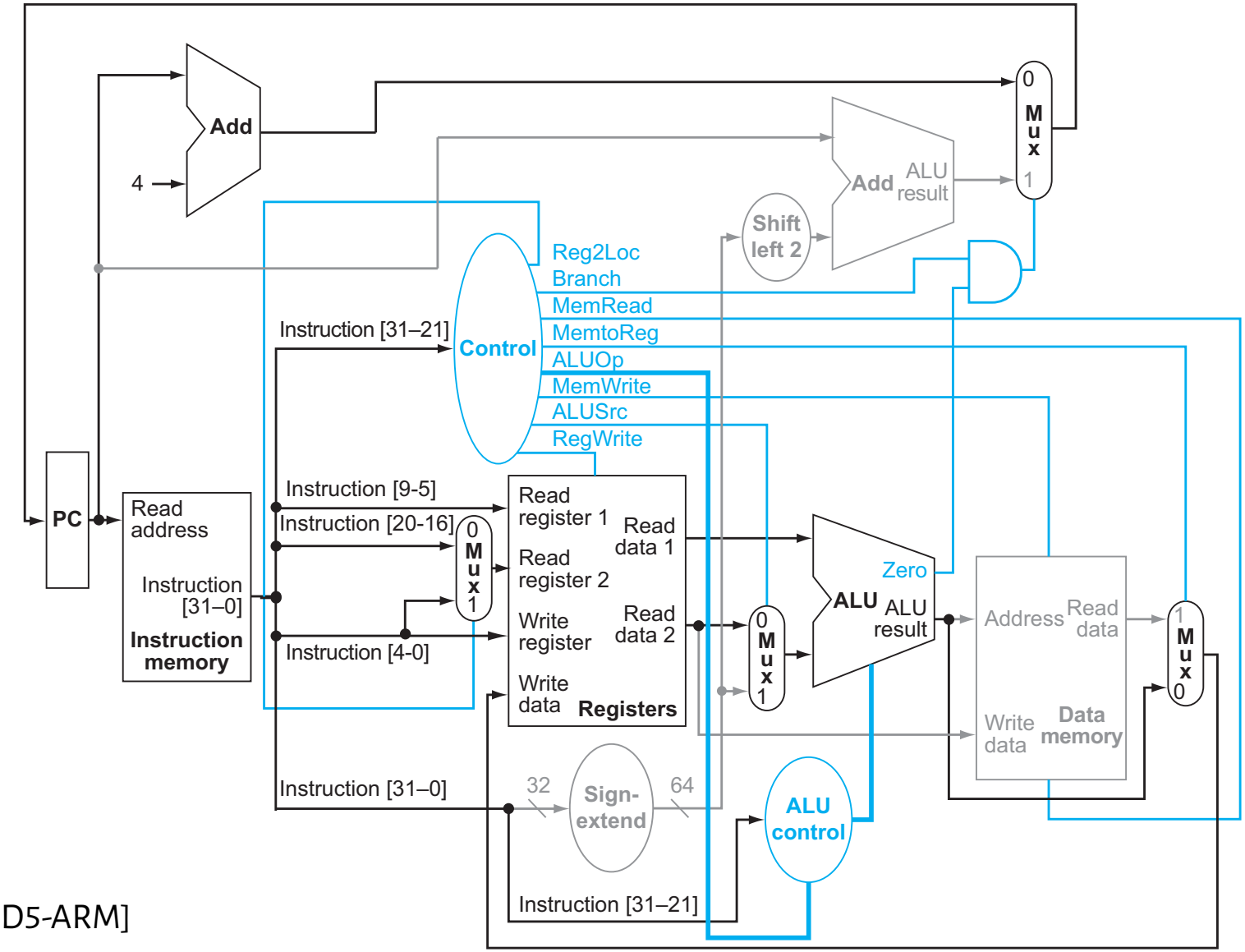
Tabela de verdade do controlador principal

Input or output	Signal name	R-format	LDUR	STUR	CBZ
Inputs	I[31]	1	1	1	1
	I[30]	X	1	1	0
	I[29]	X	1	1	1
	I[28]	0	1	1	1
	I[27]	1	1	1	0
	I[26]	0	0	0	1
	I[25]	1	0	0	0
	I[24]	X	0	0	0
	I[23]	0	0	0	X
	I[22]	0	1	0	X
	I[21]	0	0	0	X
Outputs	Reg2Loc	0	X	1	1
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Fonte: [COD5-ARM]

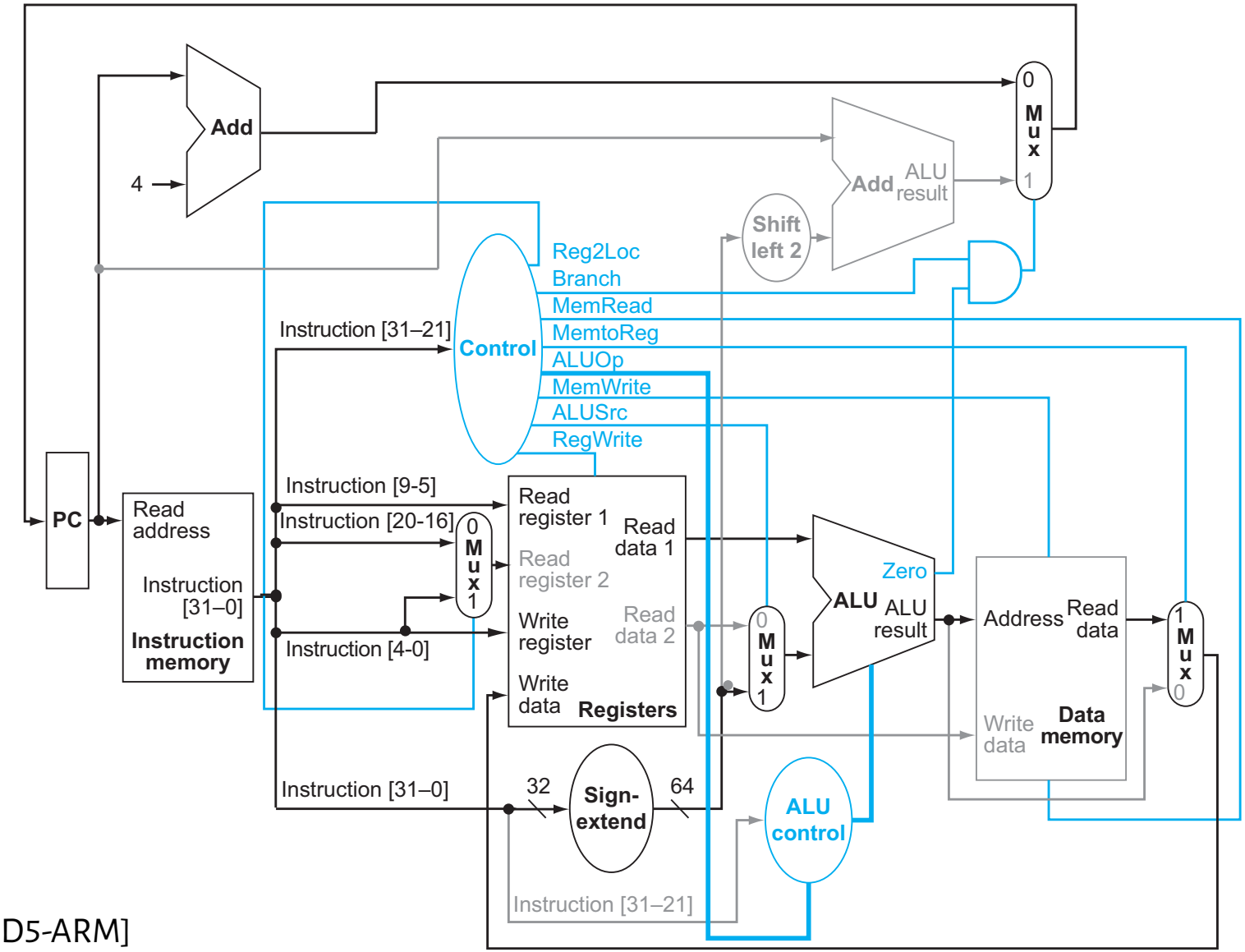
Situação durante o tratamento de instruções do tipo R

(Sinais e componentes a cinzento não estão activos.)



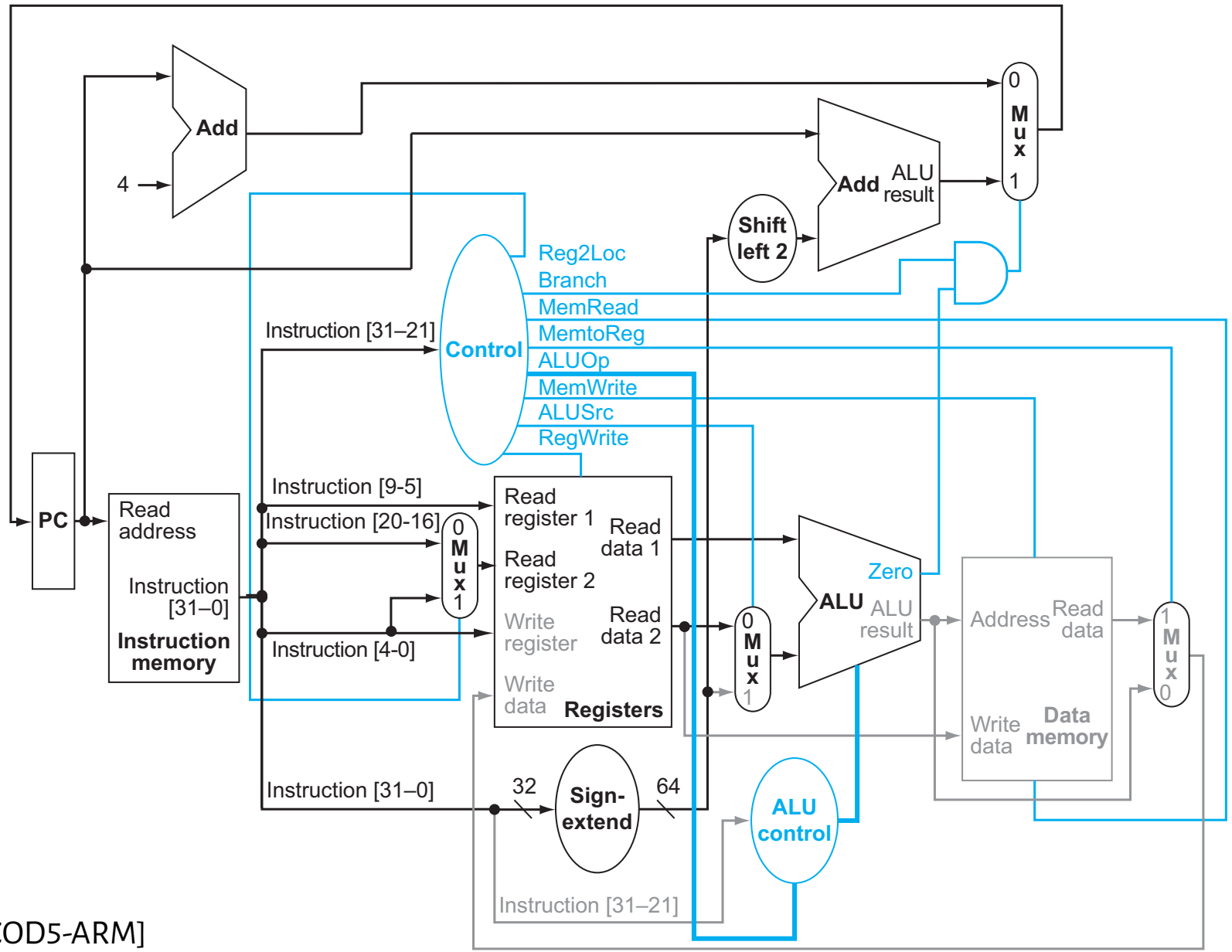
Fonte: [COD5-ARM]

Situação durante o tratamento de instruções de load



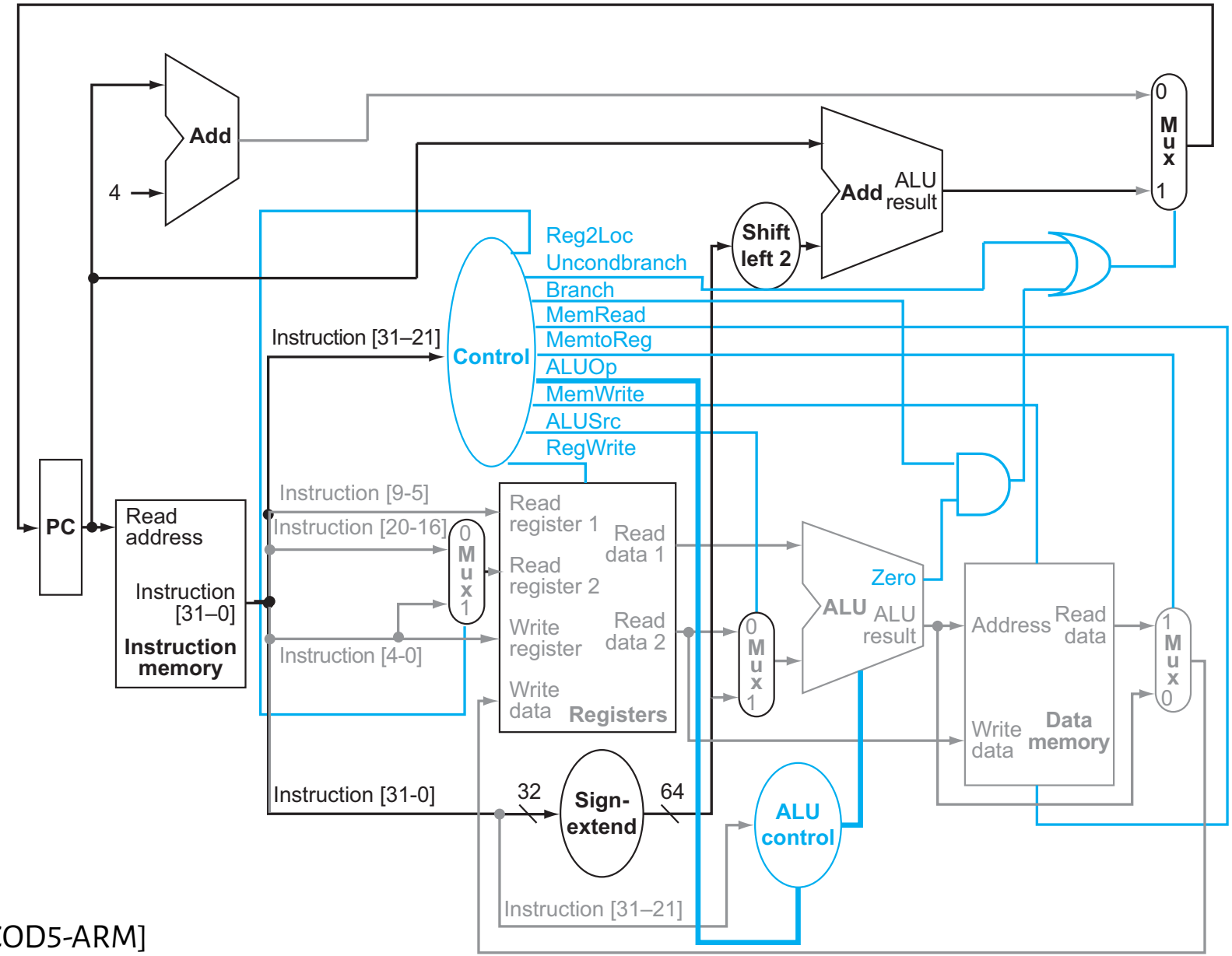
Fonte: [COD5-ARM]

Situação durante o tratamento do salto condicional



Fonte: [COD5-ARM]

Suporte para instrução de salto incondicional



Fonte: [COD5-ARM]

Limitações da implementação "uniciclo"

- Unidades de controlo "uniciclo" não são usadas em processadores atuais, porque não beneficiam o desempenho.
- O período de relógio é determinado pela instrução mais demorada (provavelmente a instrução de `load`, neste caso).
- Esta abordagem gasta **um ciclo por instrução**, mas o **período de relógio** pode ser longo.
- Para CPUs muito simples, pode ser um abordagem viável.
- Para CPUs com instruções de complexidade muito diferente não é uma boa abordagem.
  - Divisão inteira é muito mais demorada que adição;
  - Operações de vírgula flutuante são muito mais demoradas que operações com inteiros.
- Esta abordagem não permite seguir a regra prática:

  Reduzir o tempo de processamento dos casos mais comuns
- Alternativa: dividir tratamento de instruções em etapas e usar um período de relógio mais curto (um ciclo por etapa).

Referências

COD5-ARM D. A. Patterson & J. L. Hennessey, Computer Organization and Design: The ARM Edition

Os tópicos tratados nesta apresentação são descritos nas seguintes secções de [COD5-ARM]:

- 4.1–4.4