

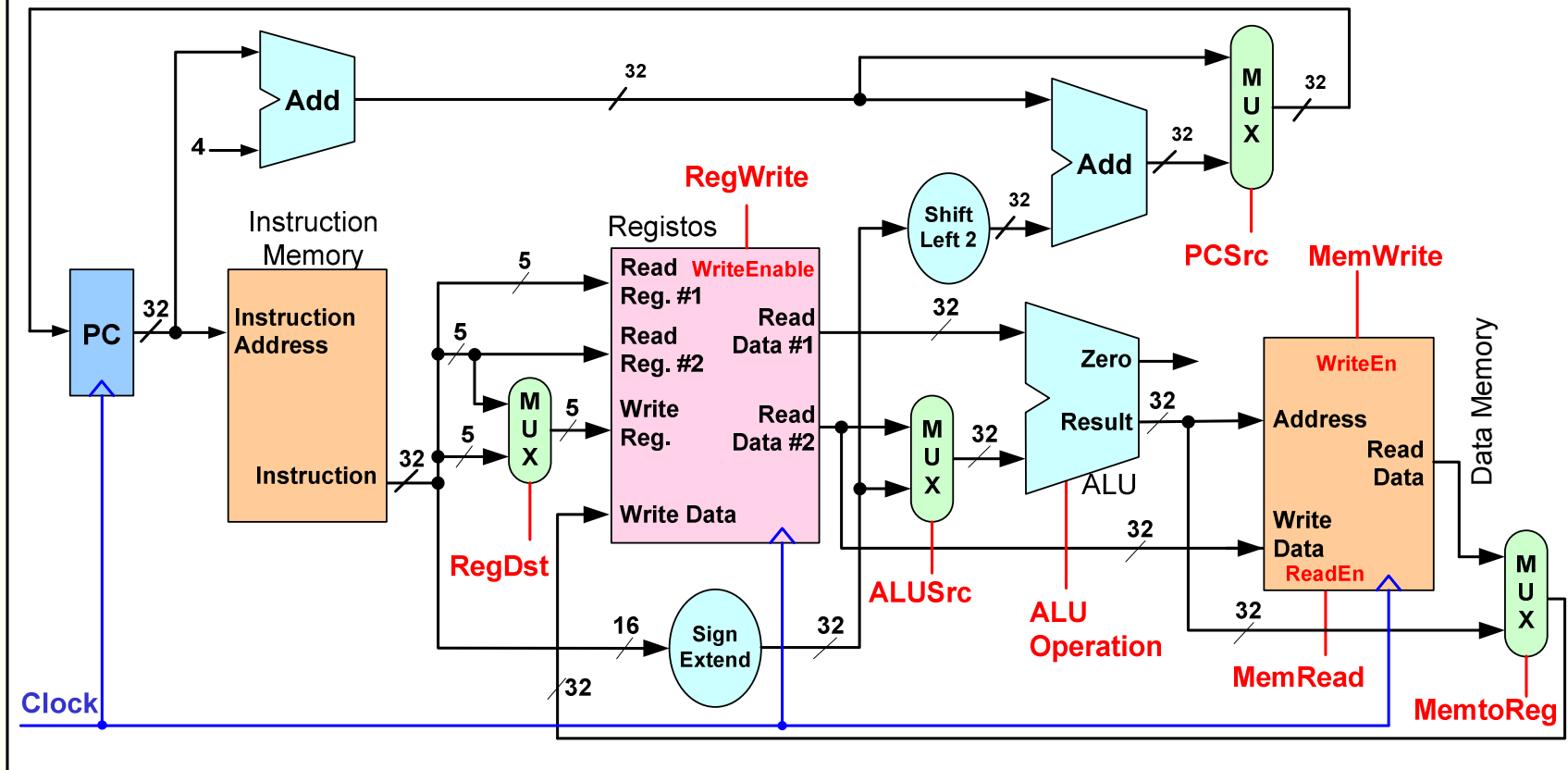
Aulas 14 e 15

- A unidade de controlo principal do *datapath single-cycle*
- A unidade de controlo da ALU
- Desenho das unidades de controlo do *datapath* e da ALU
- Exemplos de funcionamento do *datapath* com unidade de controlo
- Suporte para a instrução *jump* (j)

José Luís Azevedo, Bernardo Cunha, Arnaldo Oliveira, Pedro Lavrador

Datapath – unidade de controlo

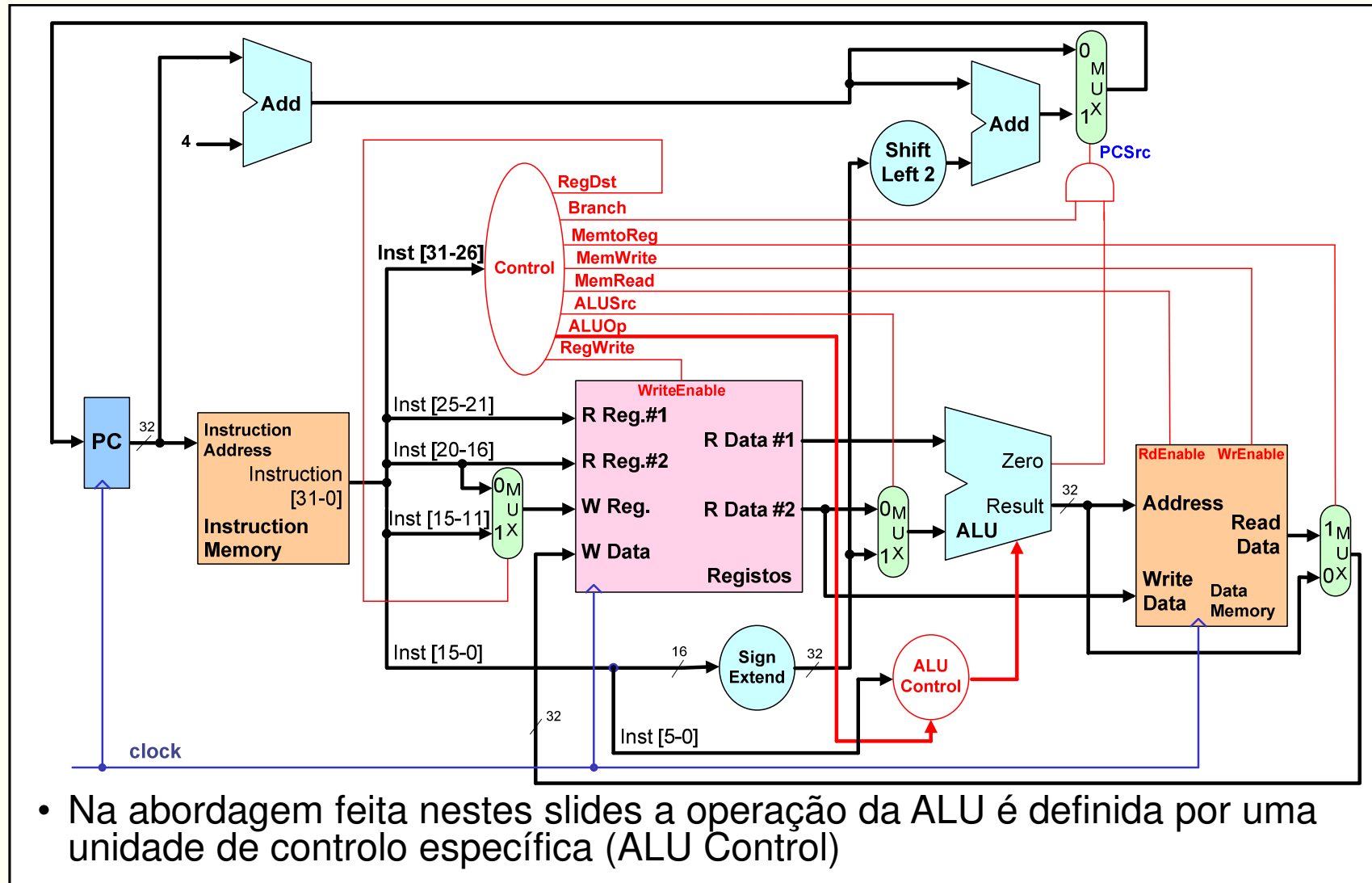
- A unidade de controlo deve gerar os sinais de controlo (identificados a vermelho) para: 1) elementos de estado: banco de registos e memória de dados; 2) *multiplexers* e ALU



Datapath – unidade de controlo

- Alguns dos elementos de estado presentes no *datapath* são acedidos em todos os ciclos de relógio (PC e memória de instruções). Nestes casos não há necessidade de explicitar um sinal de controlo
- Outros elementos de estado podem ser lidos ou escritos dependendo da instrução que estiver a ser executada (memória de dados e banco de registos). Para estes é necessário explicitar os respetivos sinais de controlo
- Para a ALU e para os elementos combinatórios que fazem o encaminhamento da informação (*multiplexers*) também é necessário definir os respetivos sinais de controlo
- A **escrita** nos elementos de estado é sempre realizada de forma síncrona

Datapath – unidade de controlo



Unidade de controlo da ALU

- As instruções básicas que fazem uso da ALU são:
 - **Load e store** – para calcular o endereço da memória externa
 - **Branch if equal / not equal** – para determinar se os operandos são iguais ou diferentes
 - **Aritméticas e lógicas** – para efetuar a respetiva operação
- A operação a realizar na ALU depende:
 - dos campos **opcode** e **funct** nas instruções aritméticas e lógicas de tipo R: **$ALUControl = f(opcode, funct)$**
 - do campo **opcode** nas restantes instruções:
 $ALUControl = f(opcode)$
- Assim, a geração dos sinais de controlo da ALU pode ser realizada em dois níveis:
 - Nível 1: **$ALUOp = g(opcode)$**
 - Nível 2: **$ALUControl = f(ALUOp, funct)$**

Unidade de controlo da ALU

- A relação entre o tipo de instruções, o campo “funct”, a operação efetuada pela ALU e os sinais de controlo da mesma, pode ser resumida pela seguinte tabela

ALU Control	ALU Action
0 0 0	And
0 0 1	Or
0 1 0	Add
1 1 0	Subtract
1 1 1	Set if Less Than

Instruction	OpCode	Funct	ALU Action	ALUOp	ALU Control
load word	100011 ("lw")	xxxxxx	add	00	010
store word	101011 ("sw")	xxxxxx	add	00	010
addi	001000 ("addi")	xxxxxx	add	00	010
branch if equal	000100 ("beq")	xxxxxx	subtract	01	110
add	000000 (R-Type)	100000	add	10	010
subtract	000000 (R-Type)	100010	subtract	10	110
and	000000 (R-Type)	100100	and	10	000
or	000000 (R-Type)	100101	or	10	001
set if less than	000000 (R-Type)	101010	set if less than	10	111
set if less than imm	001010 ("slti")	xxxxxx	set if less than	11	111

Unidade de controlo da ALU

```
library ieee;
use ieee.std_logic_1164.all;

entity ALUControlUnit is
    port (ALUop      : in  std_logic_vector(1 downto 0);
          funct      : in  std_logic_vector(5 downto 0);
          ALUcontrol : out std_logic_vector(2 downto 0));
end ALUControlUnit;
```

Unidade de controlo da ALU

```
architecture Behavioral of ALUControlUnit is
begin
  process(ALUOp, funct)
  begin
    case ALUOp is
      when "00" => -- LW, SW, ADDI
        ALUcontrol <= "010";
      when "01" => -- BEQ
        ALUcontrol <= "110";
      when "10" => -- R-Type instructions
        case funct is
          when "100000" => ALUcontrol <= "010"; -- ADD
          when "100010" => ALUcontrol <= "110"; -- SUB
          when "100100" => ALUcontrol <= "000"; -- AND
          when "100101" => ALUcontrol <= "001"; -- OR
          when "101010" => ALUcontrol <= "111"; -- SLT
          when others => ALUcontrol <= "010";
        end case;
      when "11" => -- SLTI
        ALUcontrol <= "111";
    end case;
  end process;
end Behavioral;
```

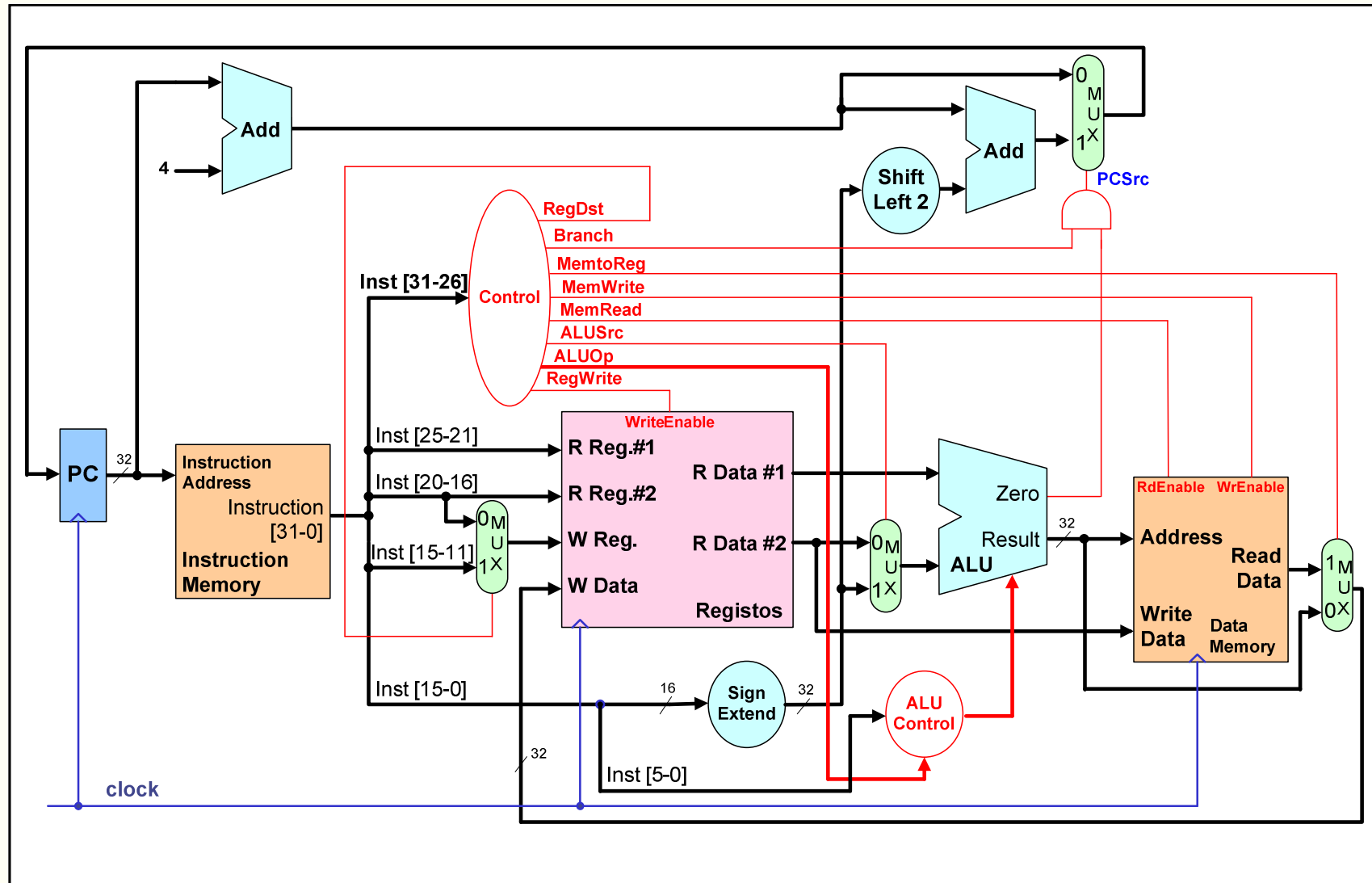
ALU Control	ALU Action
0 0 0	And
0 0 1	Or
0 1 0	Add
1 1 0	Subtract
1 1 1	Set if Less Than

ALUOp	ALU Action
00	Add
01	Subtract
10	R-Type
11	Set if Less Than

Unidade de controlo principal

- O desenho da unidade de controlo principal do nosso CPU simplificado apoia-se na observação de um conjunto de factos que decorrem da forma como são codificadas as instruções do MIPS:
 - O campo **op** (Operation Code) está situado nos bits **31-26 de todas as instruções**
 - Os índices dos 2 registos que devem ser lidos (nas instruções em que tal se aplica), surgem sempre nos bits **25-21 (rs)** e **20-16 (rt)**.
 - Nas instruções **load/store**, o **registo base de endereçamento** está sempre nos bits **25-21 (rs)**
 - As **constantes ou offsets** surgem sempre nos bits **15-0** da instrução (à excepção do “j” em que a constante surge nos bits 25-0)
 - O **registo destino** (quando se aplique) pode aparecer em um de dois campos: nos **bits 20-16** (lw, addi, slti), ou nos **bits 15-11** (instruções aritméticas e lógicas de tipo R)

Unidade de controlo principal

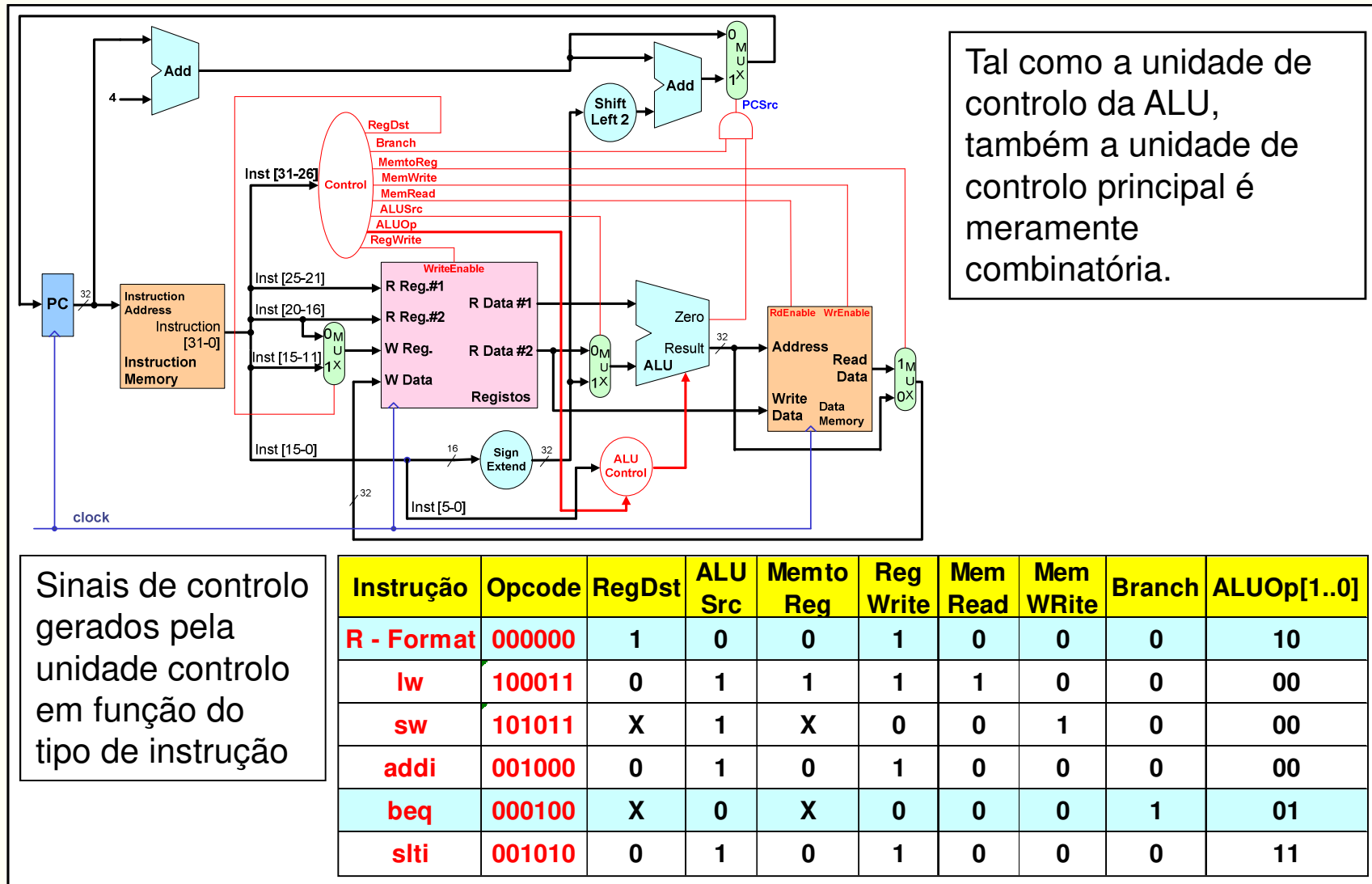


Unidade de controlo principal

- Teremos assim de especificar um total de sete (+1) sinais de controlo (para além do ALUOp). São eles:

Sinal	Efeito quando não ativo ('0')	Efeito quando ativo ('1')
MemRead	Nenhum	O conteúdo da memória de dados no endereço indicado é apresentado à saída
MemWrite	Nenhum	O conteúdo do registo de memória de dados cujo endereço é fornecido é substituído pelo valor apresentado à entrada
ALUSrc	O segundo operando da ALU provém da segunda saída do <i>File Register</i>	O segundo operando da ALU provém dos 16 bits menos significativos da instrução após extensão do sinal
RegDst	O endereço do registo destino provém do campo rt	O endereço do registo destino provém do campo rd
RegWrite	Nenhum	O registo indicado no endereço de escrita é alterado pelo valor presente na entrada de dados
MemtoReg	O valor apresentado para escrita no registo destino provém da ALU	O valor apresentado na entrada de dados dos registos internos provém da memória externa
PCSrc	O PC é substituído pelo seu valor actual mais 4	O PC é substituído pelo resultado do somador que calcula o endereço target do <i>branch</i> condicional
Branch	Nenhum	Indica que a instrução é um branch condicional

Unidade de controlo principal



Unidade de controlo principal

```
library ieee;
use ieee.std_logic_1164.all;

entity ControlUnit is
  port (OpCode      : in std_logic_vector(5 downto 0);
        RegDst      : out std_logic;
        Branch      : out std_logic;
        MemRead      : out std_logic;
        MemWrite     : out std_logic;
        MemToReg     : out std_logic;
        ALUSrc       : out std_logic;
        RegWrite     : out std_logic;
        ALUOp        : out std_logic_vector(1 downto 0));
end ControlUnit;
```

```

architecture Behavioral of ControlUnit is
begin
  process (OpCode)
  begin
    RegDst    <= '0'; Branch <= '0'; MemRead  <= '0'; MemWrite <= '0';
    MemToReg  <= '0'; ALUSrc <= '0'; RegWrite <= '0';
    ALUOp     <= "00";
    case OpCode is
      when "000000" =>    -- R-Type instructions
        ALUOp    <= "10";
        RegDst    <= '1';
        RegWrite  <= '1';
      when "000100" =>    -- BEQ
        ALUOp    <= "01";
        Branch    <= '1';
      when "100011" =>    -- LW
        ALUSrc    <= '1';
        MemToReg  <= '1';
        MemRead   <= '1';
        RegWrite  <= '1';
      when "101011" =>    -- SW
        ALUSrc    <= '1';
        MemWrite  <= '1';
      when "001000" =>    -- ADDI
        ALUSrc    <= '1';
        RegWrite  <= '1';
      when "001010" =>    -- SLTI
        ALUOp     <= "11";
        ALUSrc    <= '1';
        RegWrite  <= '1';
      when others =>
    end case;
  end process;
end Behavioral;

```

Análise do funcionamento do *datapath*

- A execução de qualquer uma das instruções suportadas ocorre no intervalo de tempo correspondente a um único ciclo de relógio: tem início numa transição ativa do relógio e termina na transição ativa seguinte
- Para simplificar a análise podemos, no entanto, admitir que a utilização dos vários elementos operativos é “sequencial” e decorre ao longo de um conjunto de operações que culminam com:
 - escrita no Banco de Registos: instruções tipo R, LW, ADDI, SLTI
 - escrita na Memória de Dados: SW
- O *Program Counter* é sempre atualizado com:
 - endereço-alvo da instrução BEQ, se os registos forem iguais (*branch taken*), ou PC+4 se forem diferentes (*branch not taken*)
 - endereço-alvo da instrução J
 - PC+4 nas restantes instruções

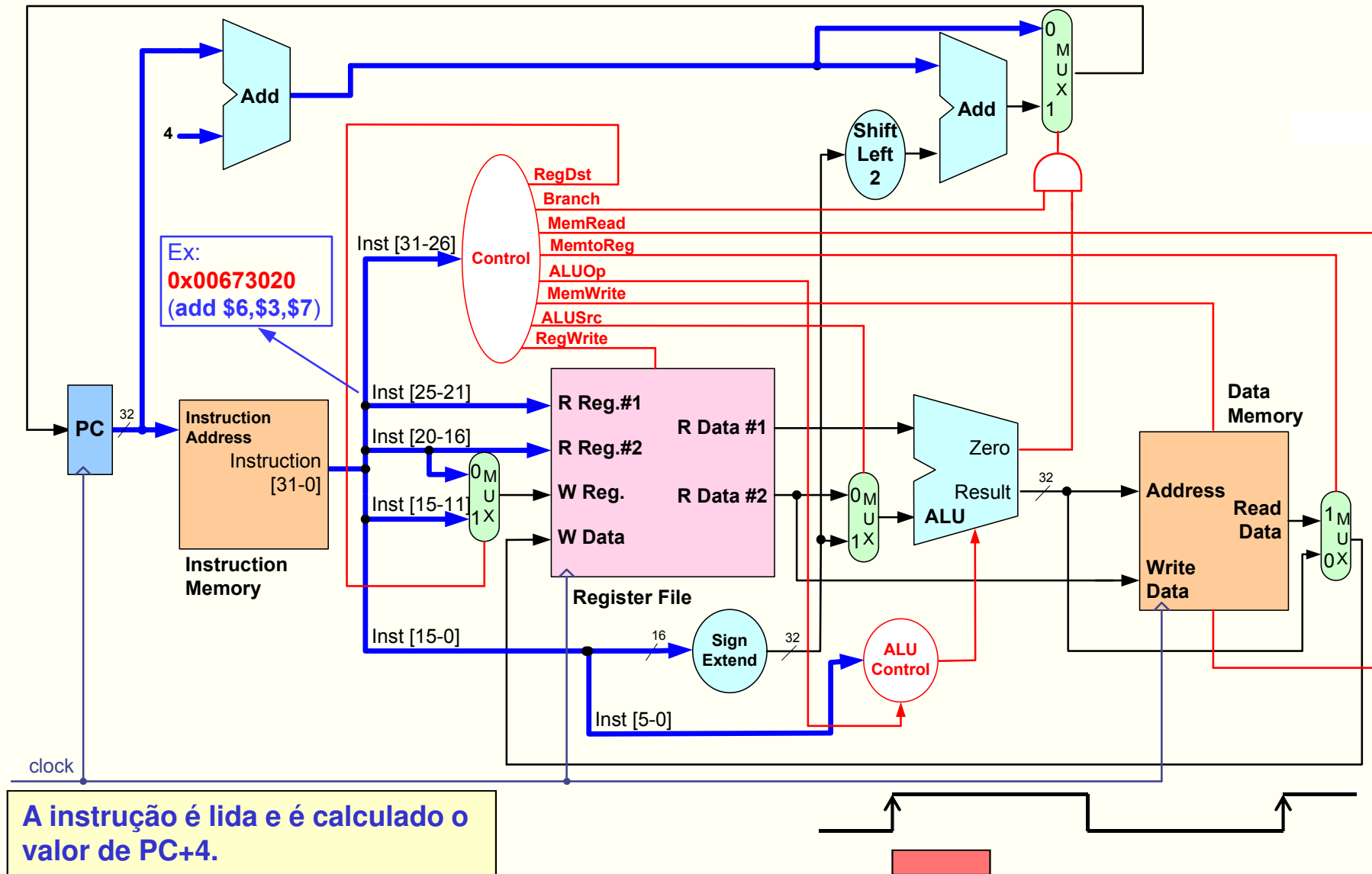
Análise do funcionamento do *datapath* – operações

- *Fetch* de uma instrução e cálculo do endereço da próxima instrução
- Leitura de dois registos do Banco de Registos
- A ALU opera sobre dois valores (a fonte dos valores a operar depende do tipo de instrução que estiver a ser executada)
- O resultado da operação efetuada na ALU:
 - é escrito no Banco de Registos (**R-Type**, **addi** e **slti**)
 - é usado como endereço para escrever na memória de dados (**sw**)
 - é usado como endereço para fazer uma leitura da memória de dados (**lw**) - o valor lido da memória de dados é depois escrito no Banco de Registos
 - é usado para decidir qual o próximo valor do PC (**beq** / **bne**)

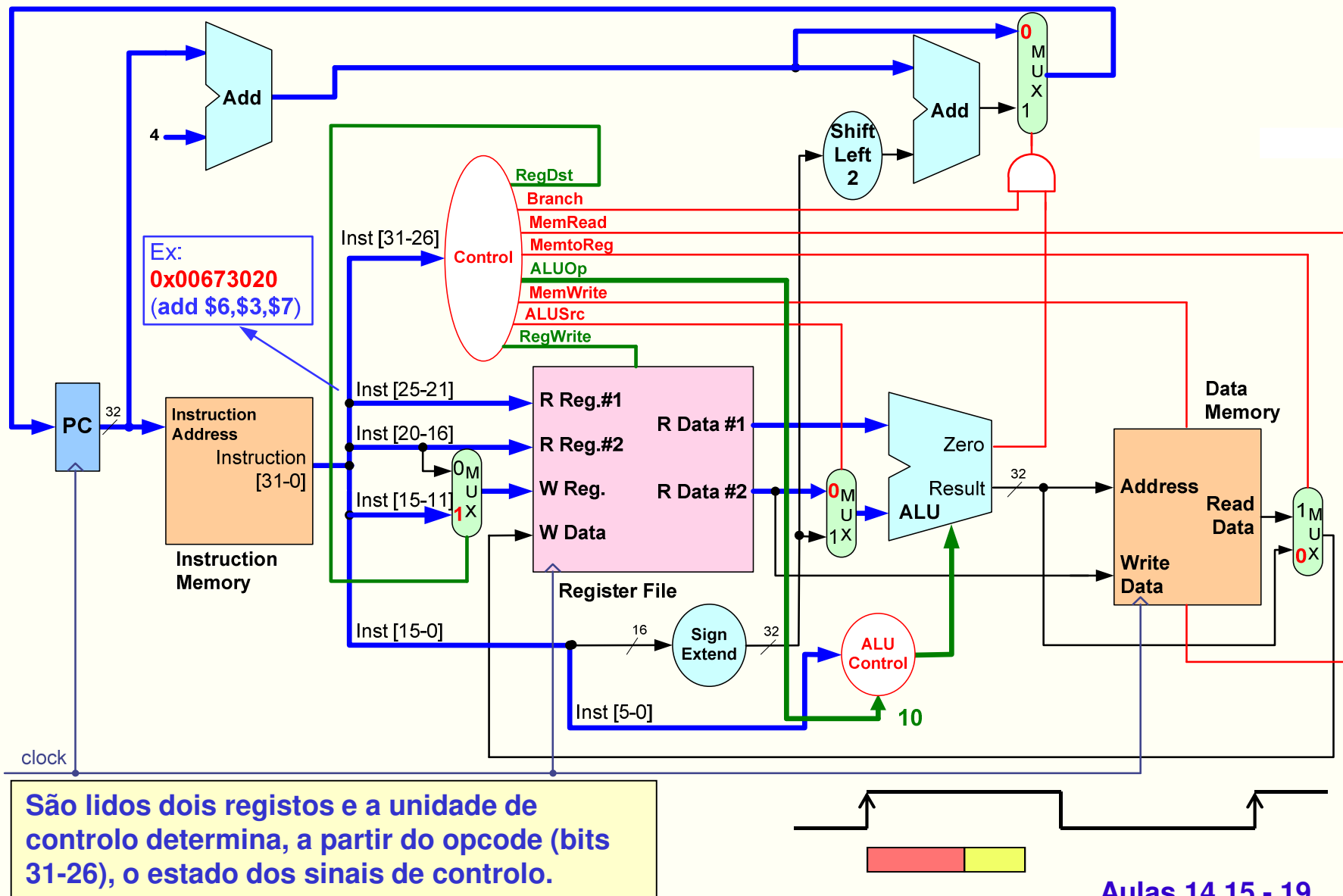
Funcionamento do *datapath* nas instruções do tipo R

- A instrução é lida e é calculado o valor de PC+4.
- São lidos dois registos e a unidade de controlo determina, a partir do *opcode* (**bits 31-26**), o estado dos sinais de controlo.
- A ALU opera sobre os dados lidos dos dois registos, de acordo com a função codificada **nos bits 5-0** da instrução.
- O resultado produzido pela ALU será escrito no registo especificado nos **bits 15-11** da instrução (rd), na próxima transição ativa do relógio.

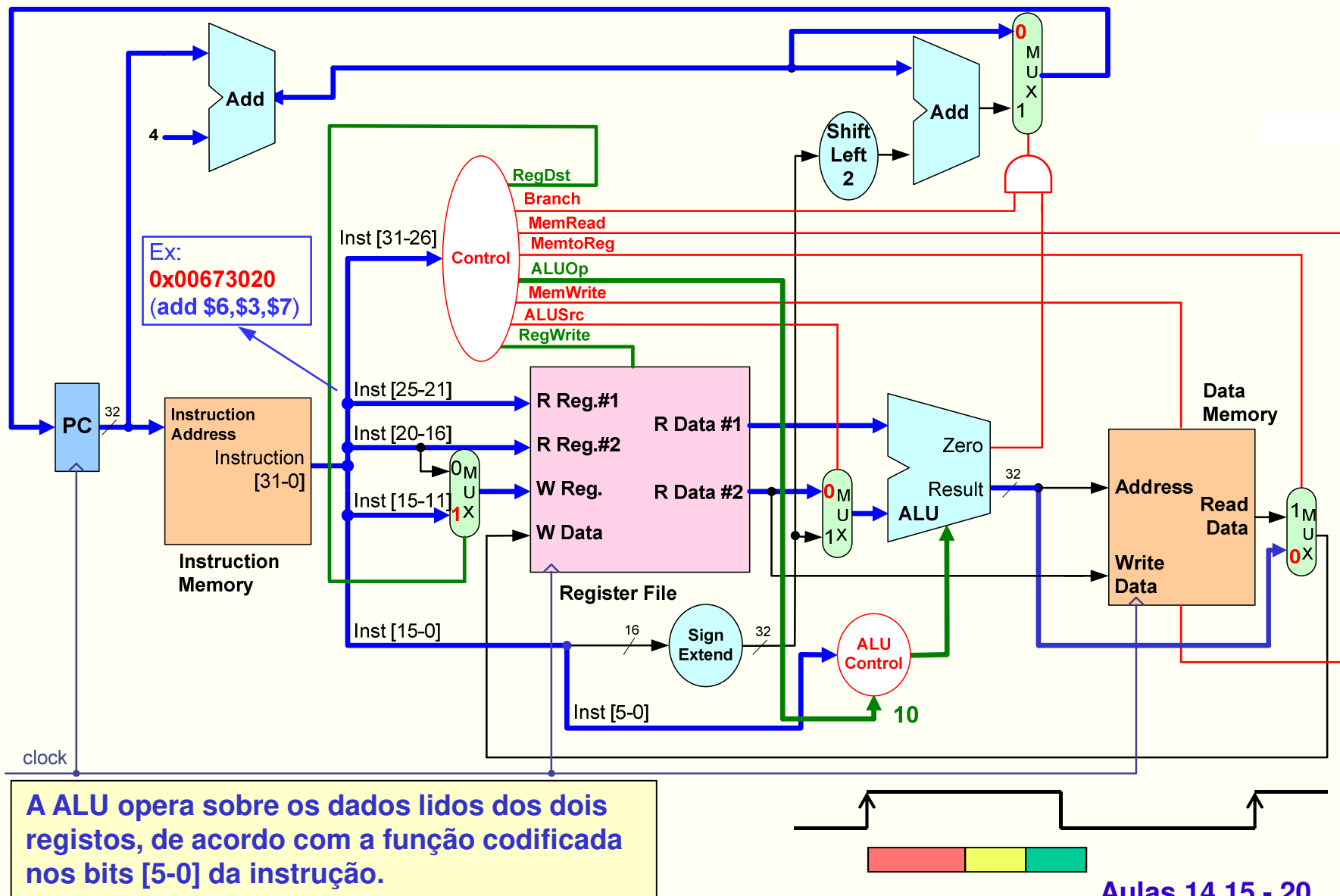
Funcionamento do *datapath* nas instruções tipo R (1)



Funcionamento do *datapath* nas instruções tipo R (2)

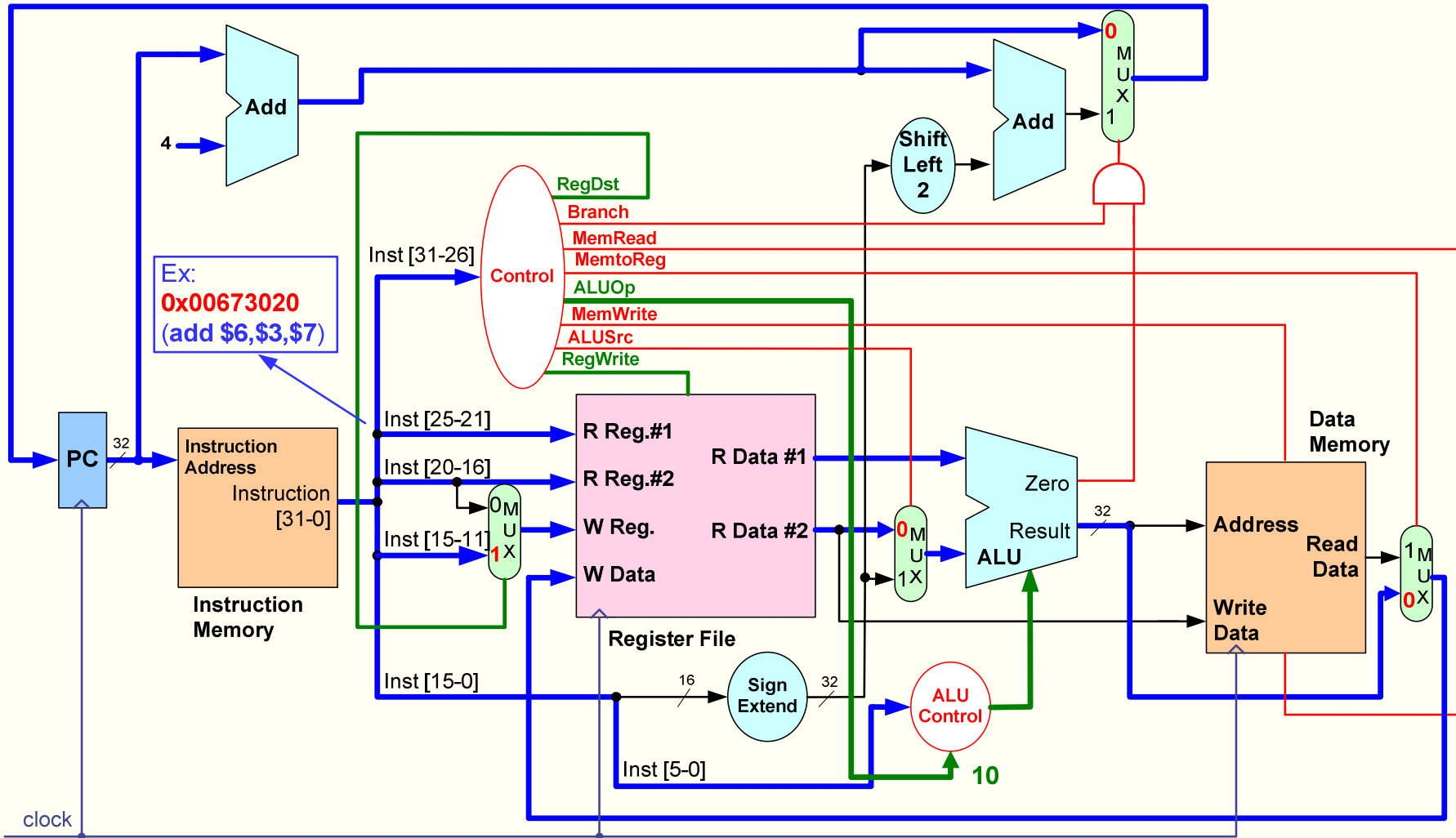


Funcionamento do *datapath* nas instruções tipo R (3)



A ALU opera sobre os dados lidos dos dois registros, de acordo com a função codificada nos bits [5-0] da instrução.

Funcionamento do *datapath* nas instruções tipo R (4)

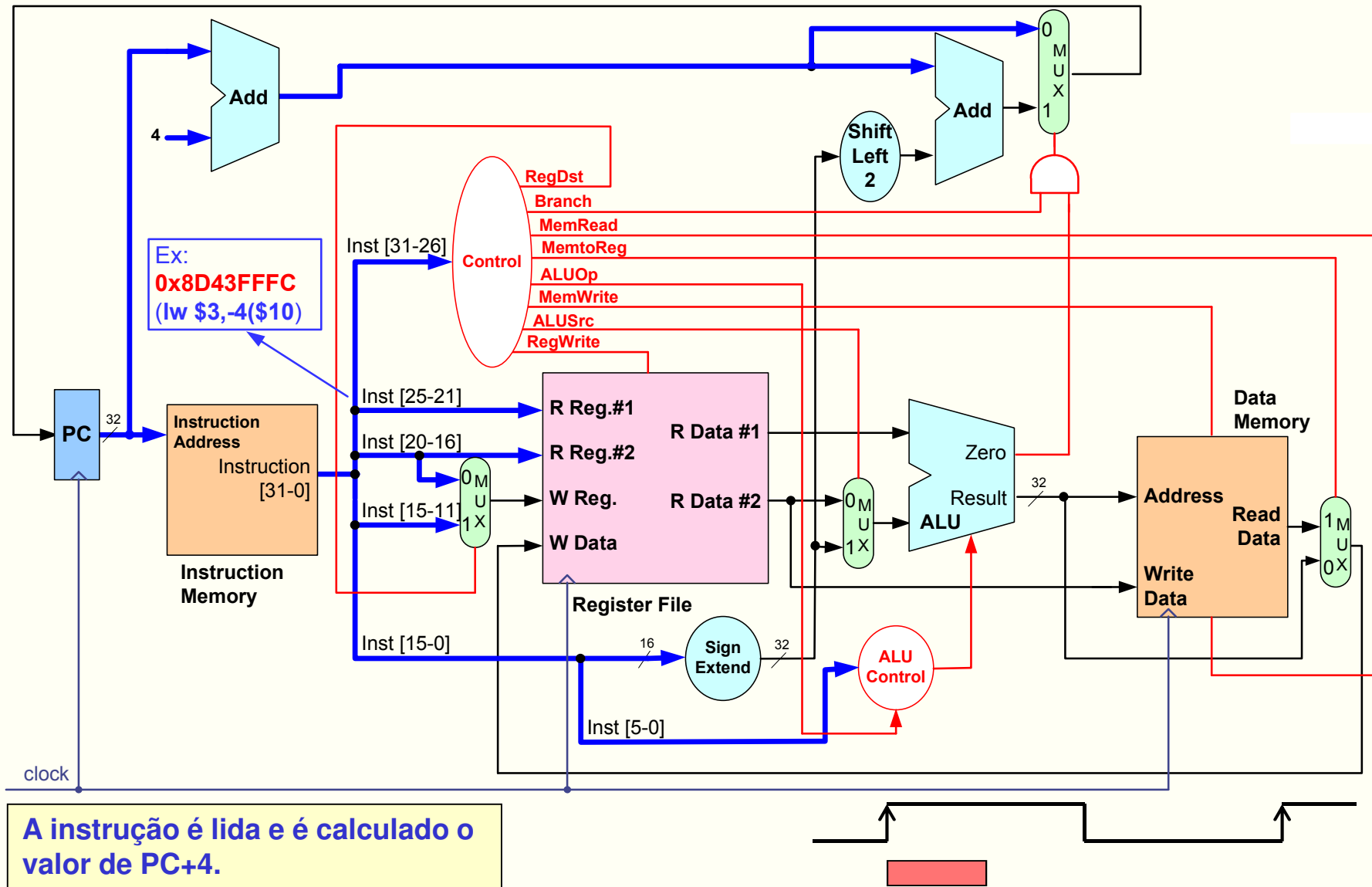


O resultado produzido pela ALU será escrito no registo especificado nos bits 15-11 da instrução (rd), na próxima transição ativa do relógio.

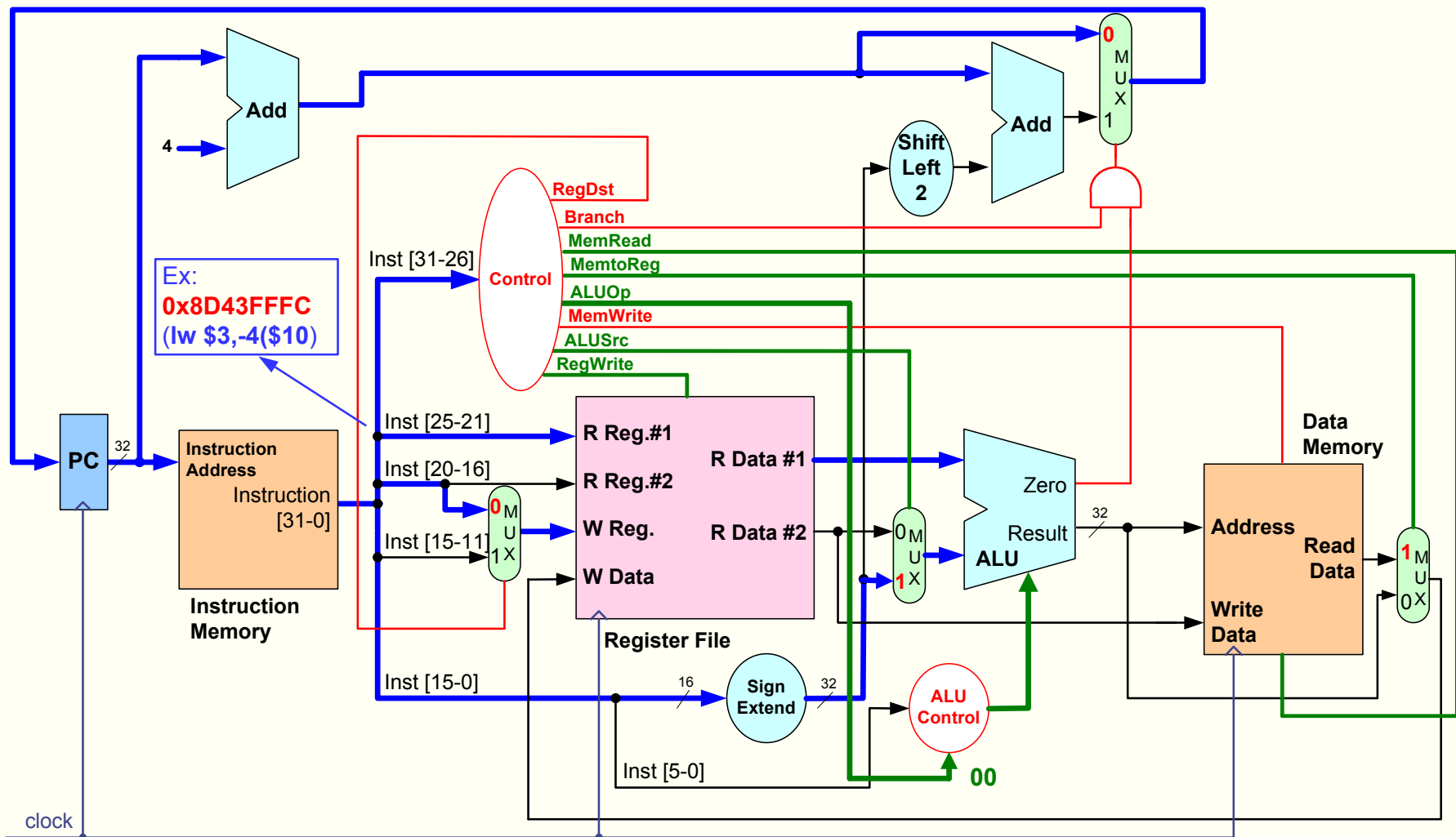
Funcionamento do *datapath* na instrução *load word* ("lw")

- A instrução é lida e é calculado o valor de PC+4.
- É lido um registo e a unidade de controlo determina, a partir do *opcode*, o estado dos sinais de controlo.
- A ALU soma o valor lido do registo especificado nos **bits 25-21** (rs) com os 16 bits (extendidos com sinal para 32) do campo *offset* da instrução (**bits 15-0**).
- O resultado produzido pela ALU constitui o endereço de acesso à memória de dados. A memória é lida nesse endereço.
- A *word* lida da memória será escrita no registo especificado nos **bits 20-16** da instrução (rt), na próxima transição ativa do relógio.

Funcionamento do *datapath* na instrução *load word* (1)

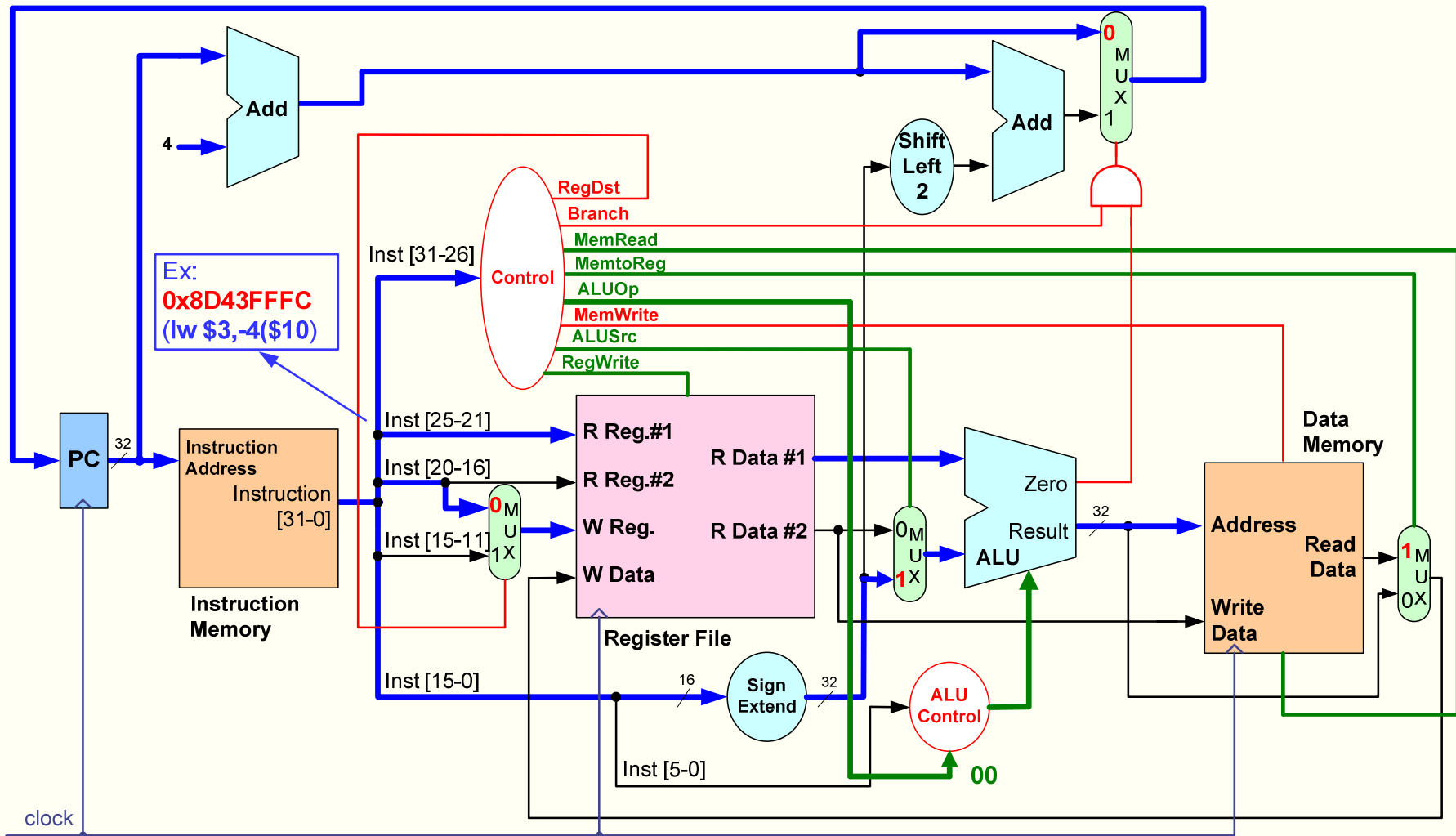


Funcionamento do *datapath* na instrução *load word* (2)



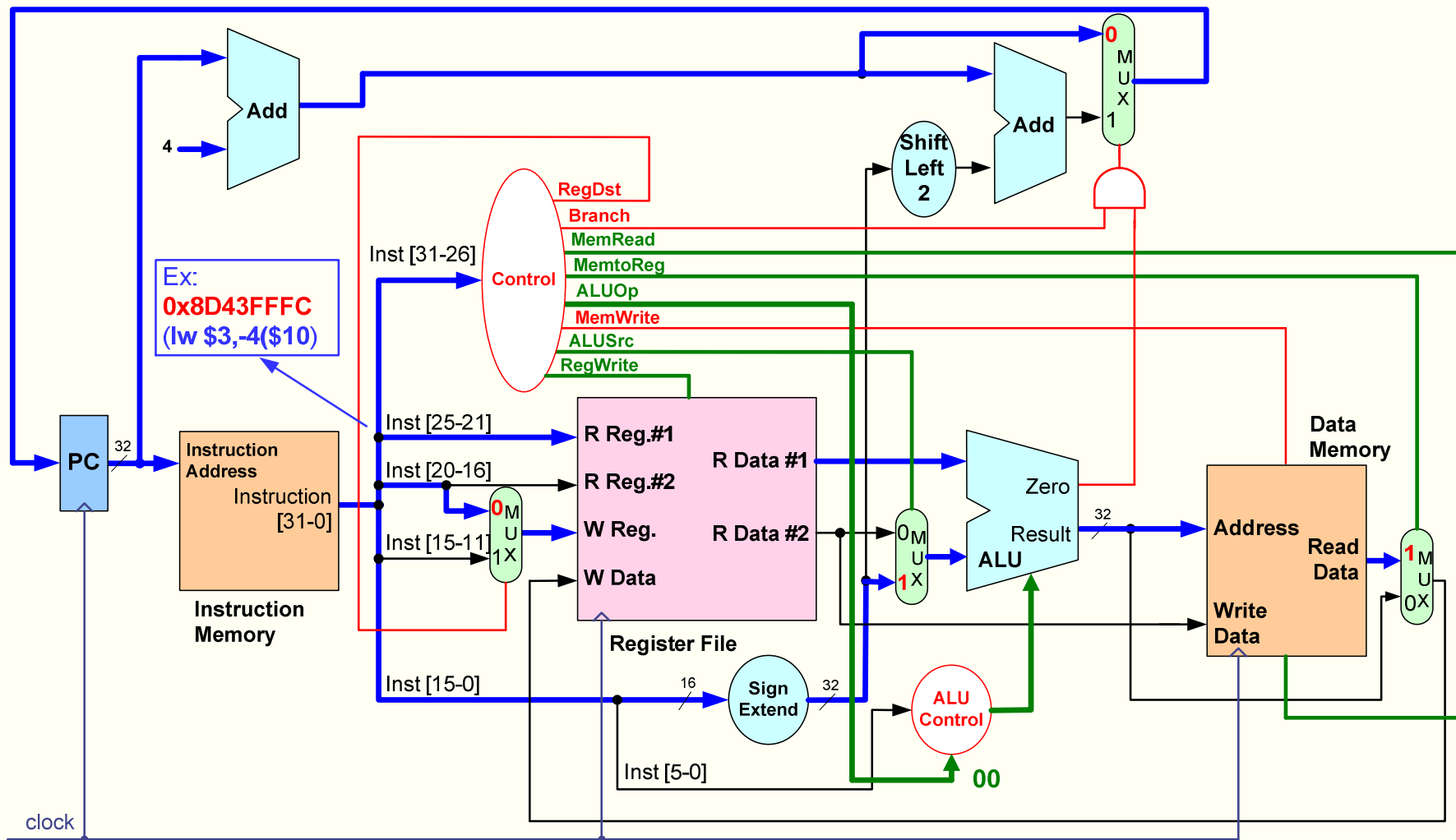
É lido um registo e a unidade de controlo determina, a partir do opcode, o estado dos sinais de controlo.

Funcionamento do *datapath* na instrução *load word* (3)

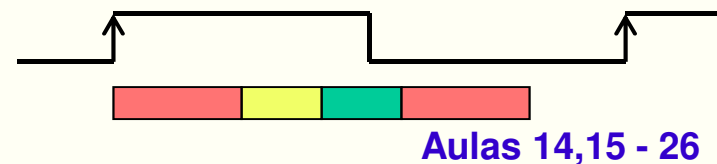


A ALU soma o valor lido do registo com os 16 bits (extendidos com sinal para 32) do campo *offset* da instrução (bits 15-0).

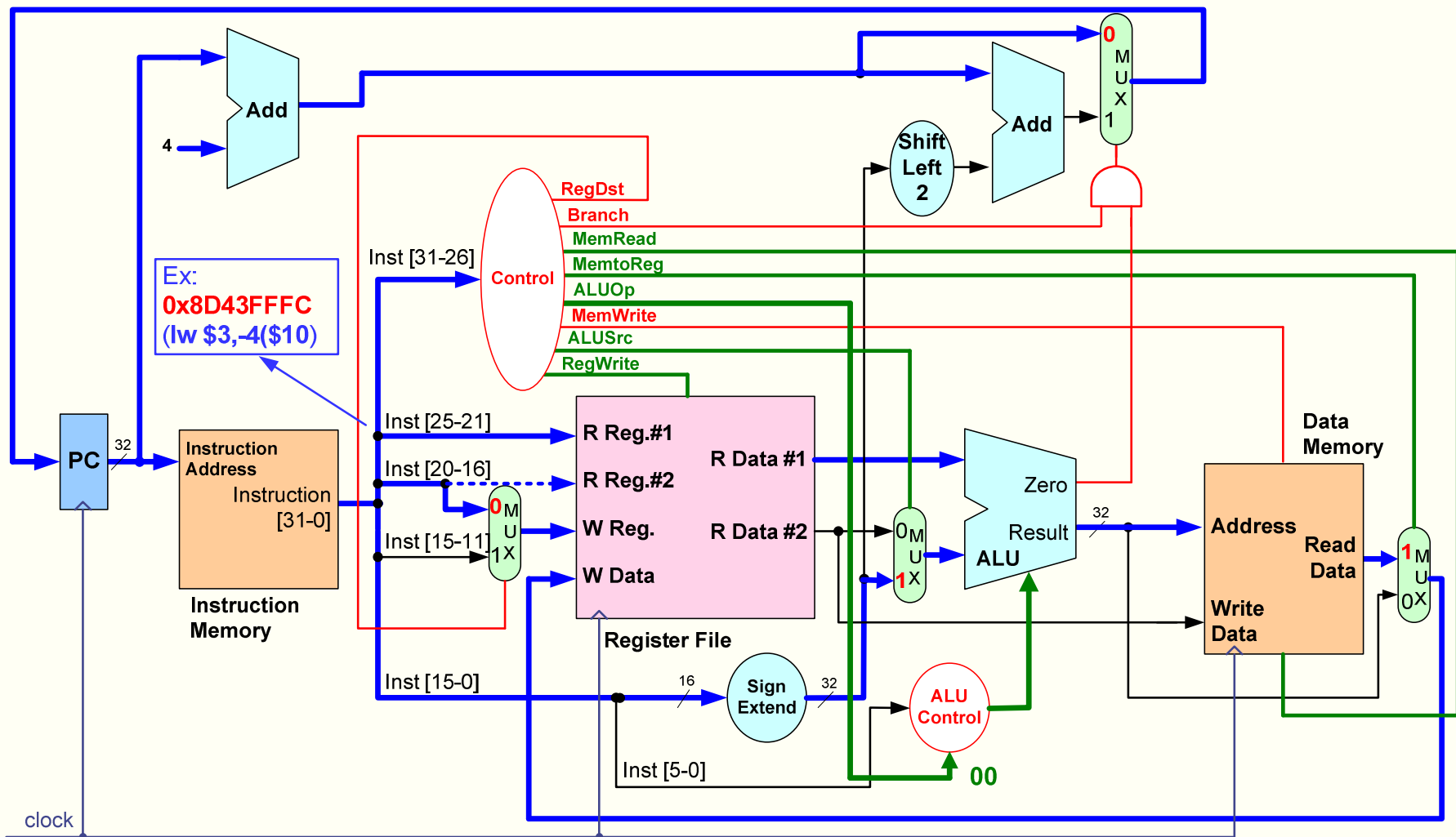
Funcionamento do *datapath* na instrução *load word* (4)



O resultado produzido pela ALU constitui o endereço de acesso à memória de dados. A memória é lida nesse endereço.



Funcionamento do *datapath* na instrução *load word* (5)

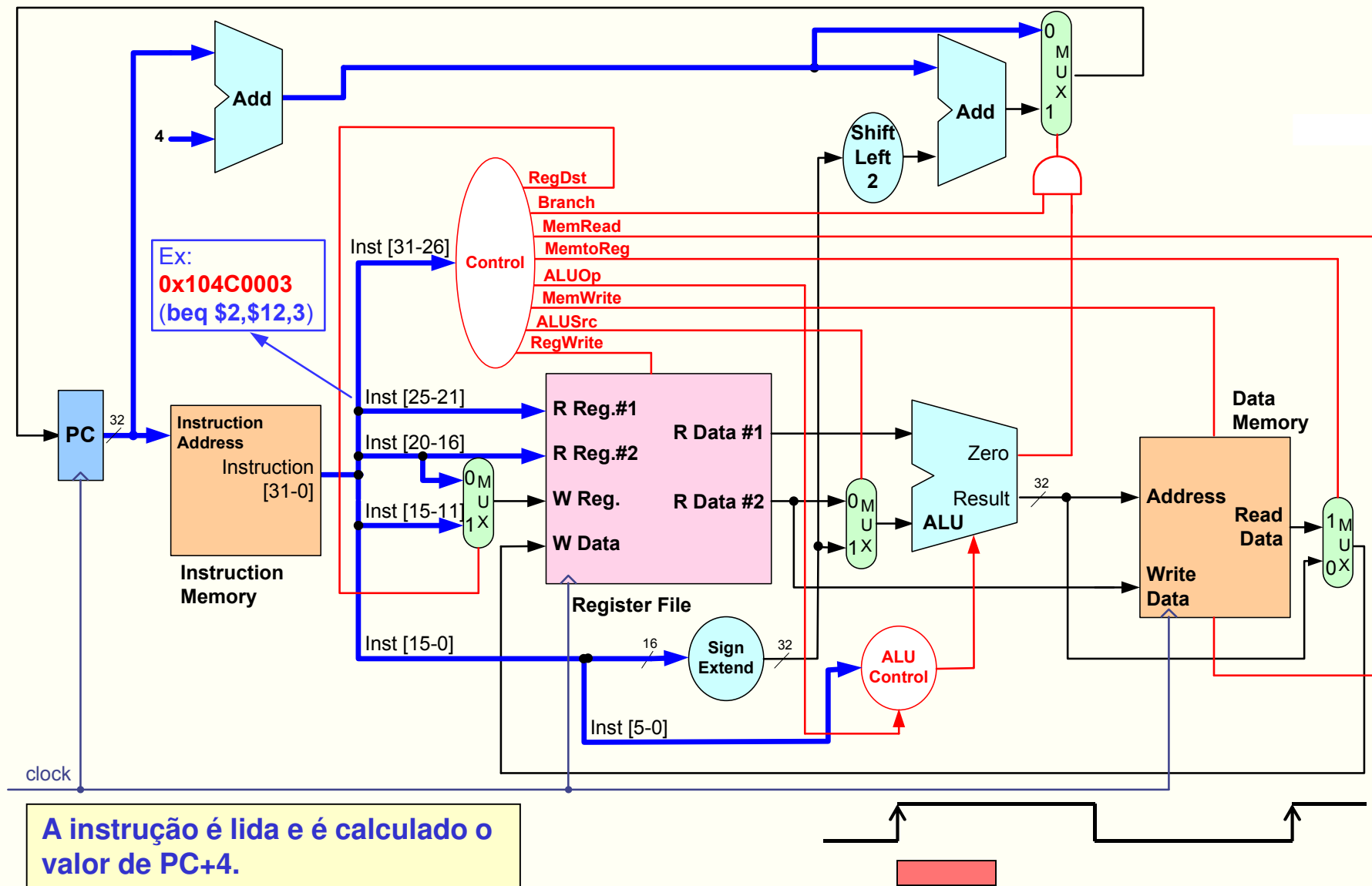


A *word* lida da memória será escrita no registo especificado nos bits 20-16 da instrução (rt), na próxima transição ativa do relógio.

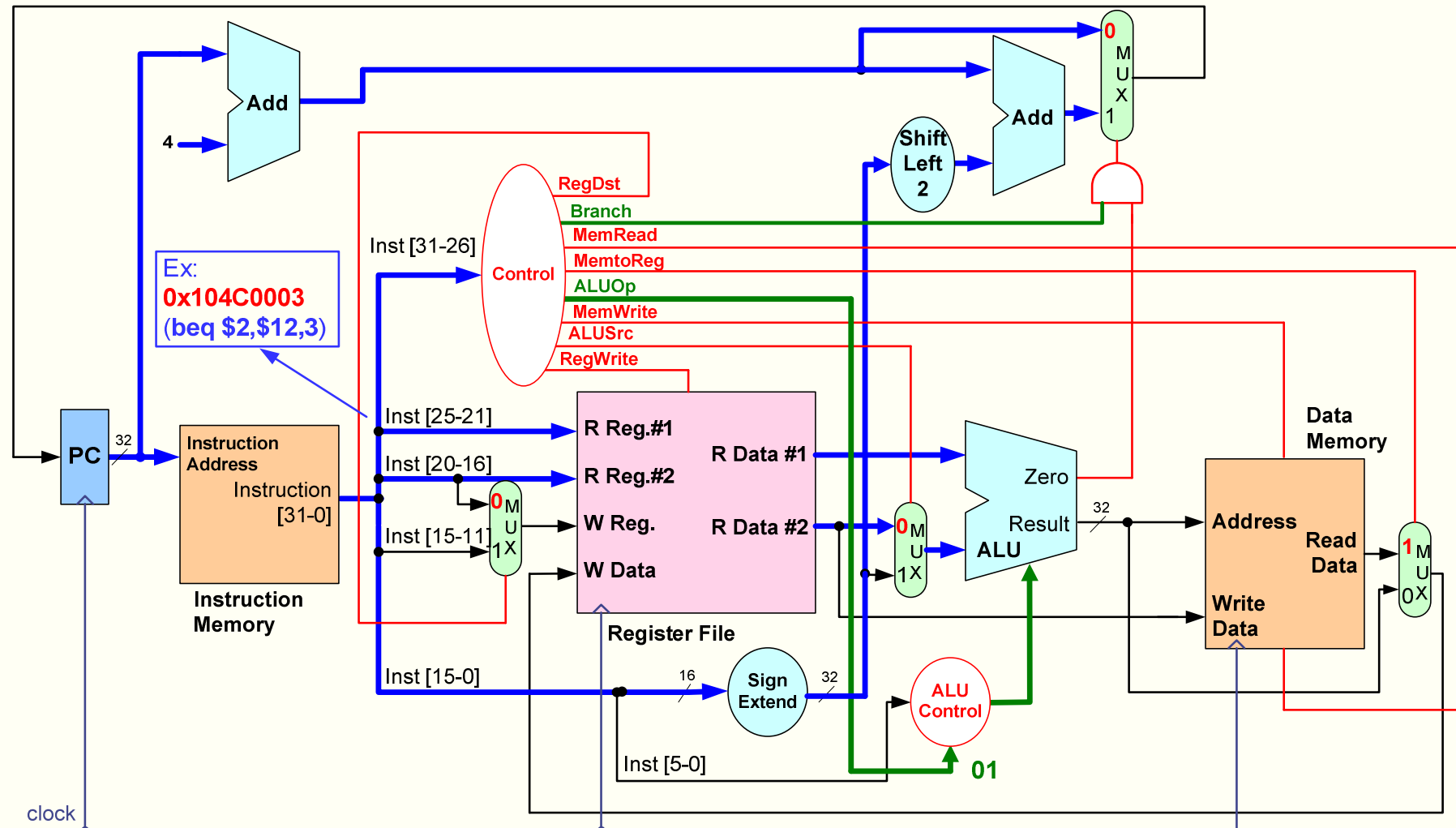
Funcionamento do *datapath* na instrução *branch if equal*

- A instrução é lida e é calculado o valor de PC+4.
- São lidos dois registos e é determinado o estado dos sinais de controlo. Os 16LSBs da instrução (sign extended x 4) são somados a PC+4 (BTA).
- A ALU faz a subtração dos dois valores lidos dos registos.
- A saída "Zero" da ALU é utilizada para decidir qual o próximo valor do PC, que será atualizado na próxima transição ativa do relógio.

Funcionamento do *datapath* na instrução "beq" (1)

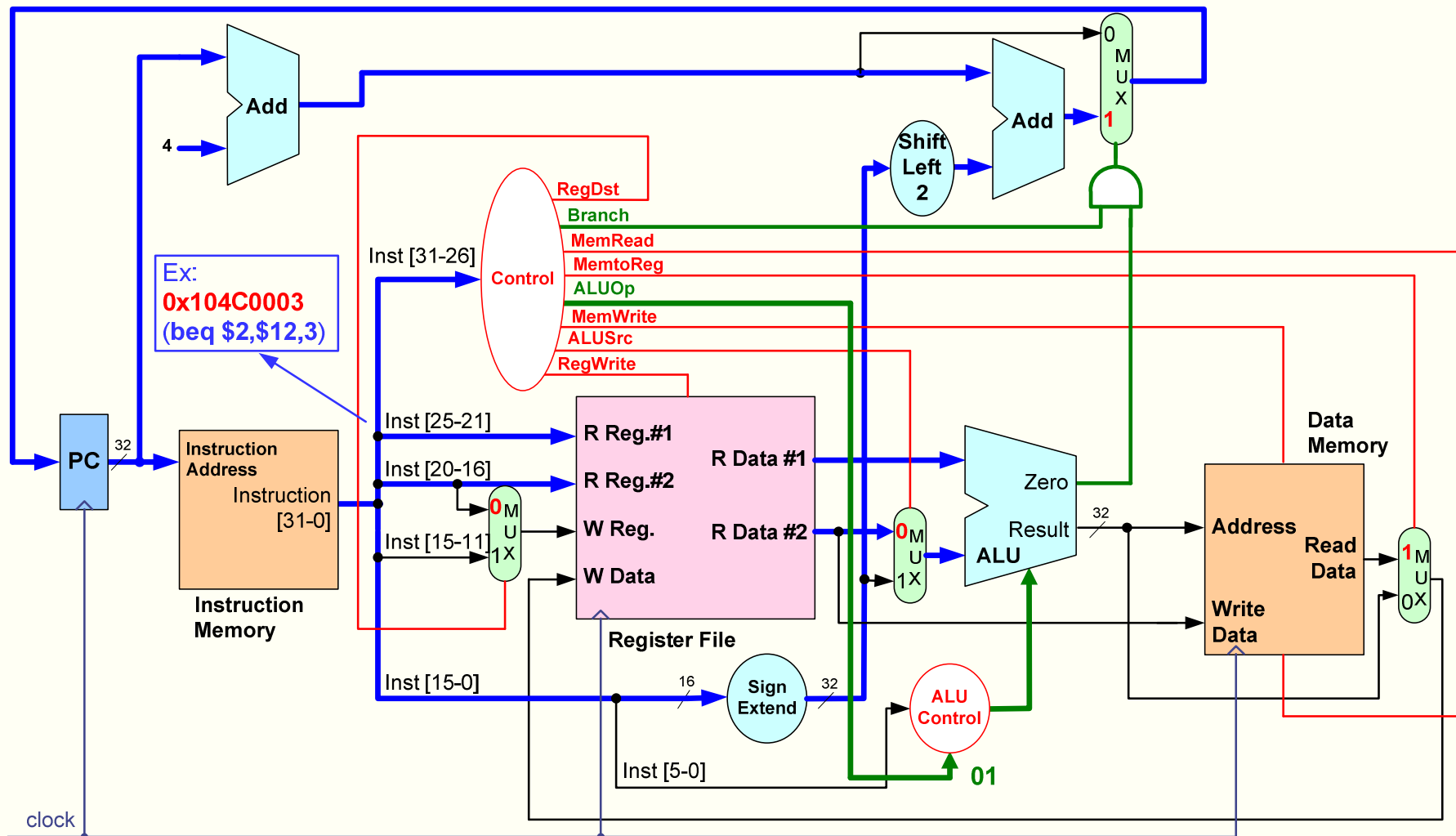


Funcionamento do *datapath* na instrução "beq" (2)



São lidos dois registros e é determinado o estado dos sinais de controlo. Os 16LSBs da instrução (sign extended x 4) são somados a PC+4.

Funcionamento do *datapath* na instrução "beq" (3)

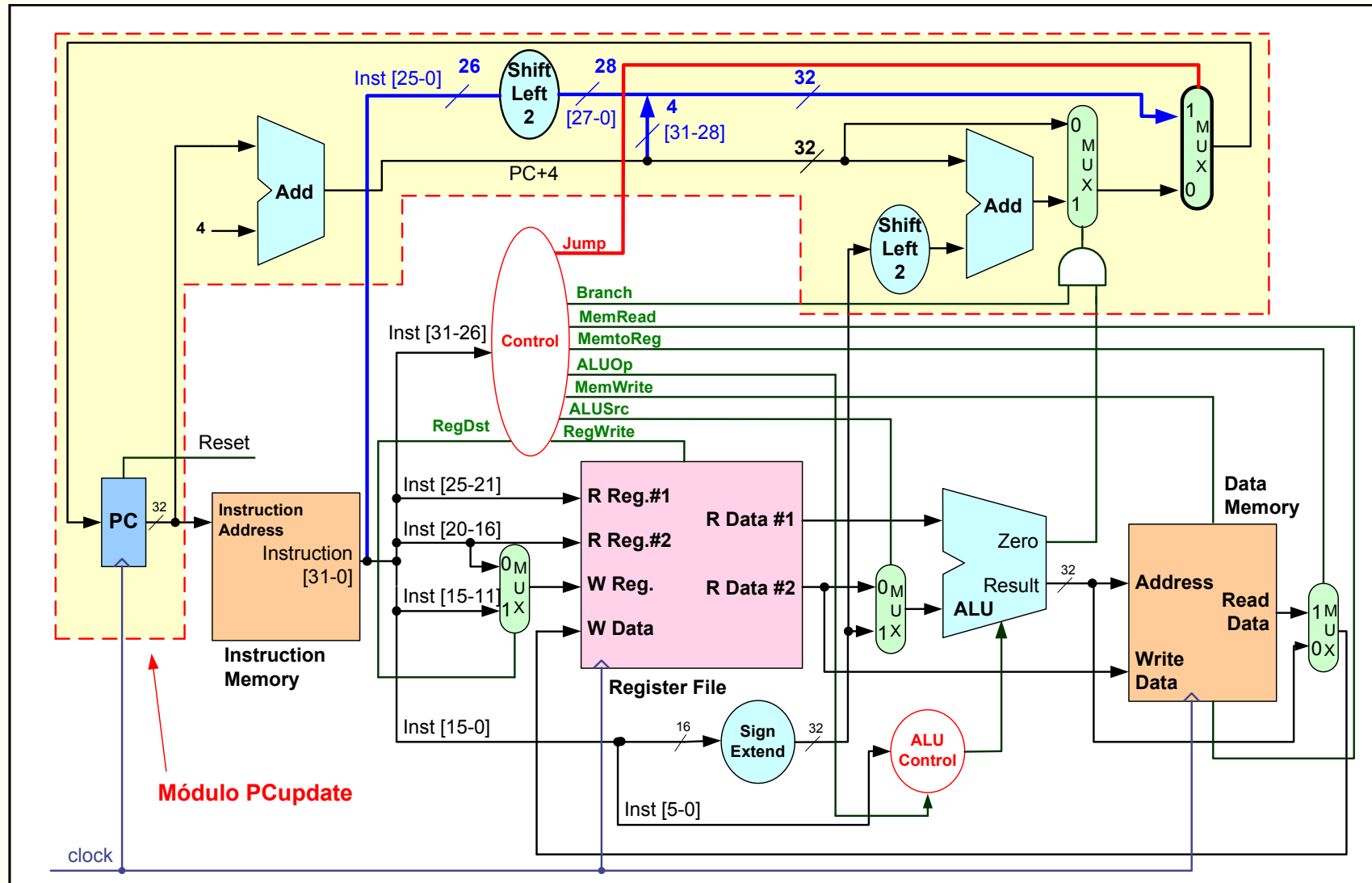


A ALU efetua a subtração dos dois valores lidos dos registos. A saída "Zero" da ALU é utilizada para decidir qual o próximo valor do PC, que será atualizado na próxima transição ativa do relógio.

Datapath com suporte para a instrução "jump"

- A instrução “jump” corresponde a um caso particular de codificação (instruções do tipo J). Nestas instruções existem apenas dois campos: o campo op (**bits 31-26**) e o campo de endereço (**bits 25-0**)
- Nas instruções de “jump”, o endereço alvo (*Jump Target Address*) obtém-se pela **concatenação** dos bits **31-28** do PC+4 com os bits do campo de endereço da instrução (26 bits) multiplicados por 4.
- Será necessário acrescentar ainda um bit de saída à unidade de controlo para seleccionar a fonte de informação disponibilizada à entrada do PC
- O *datapath* simplificado, com suporte para a instrução “j” (“jump”), fica com a configuração do slide seguinte

Datapath com suporte para a instrução “jump”



Módulo de atualização do *PC*, completo

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PCupdate is
  port (clk      : in std_logic;
        reset    : in std_logic;
        branch   : in std_logic;
        jump     : in std_logic;
        zero     : in std_logic;
        offset   : in std_logic_vector(31 downto 0);
        jAddr    : in std_logic_vector(25 downto 0);
        pc       : out std_logic_vector(31 downto 0));
end PCupdate;
```

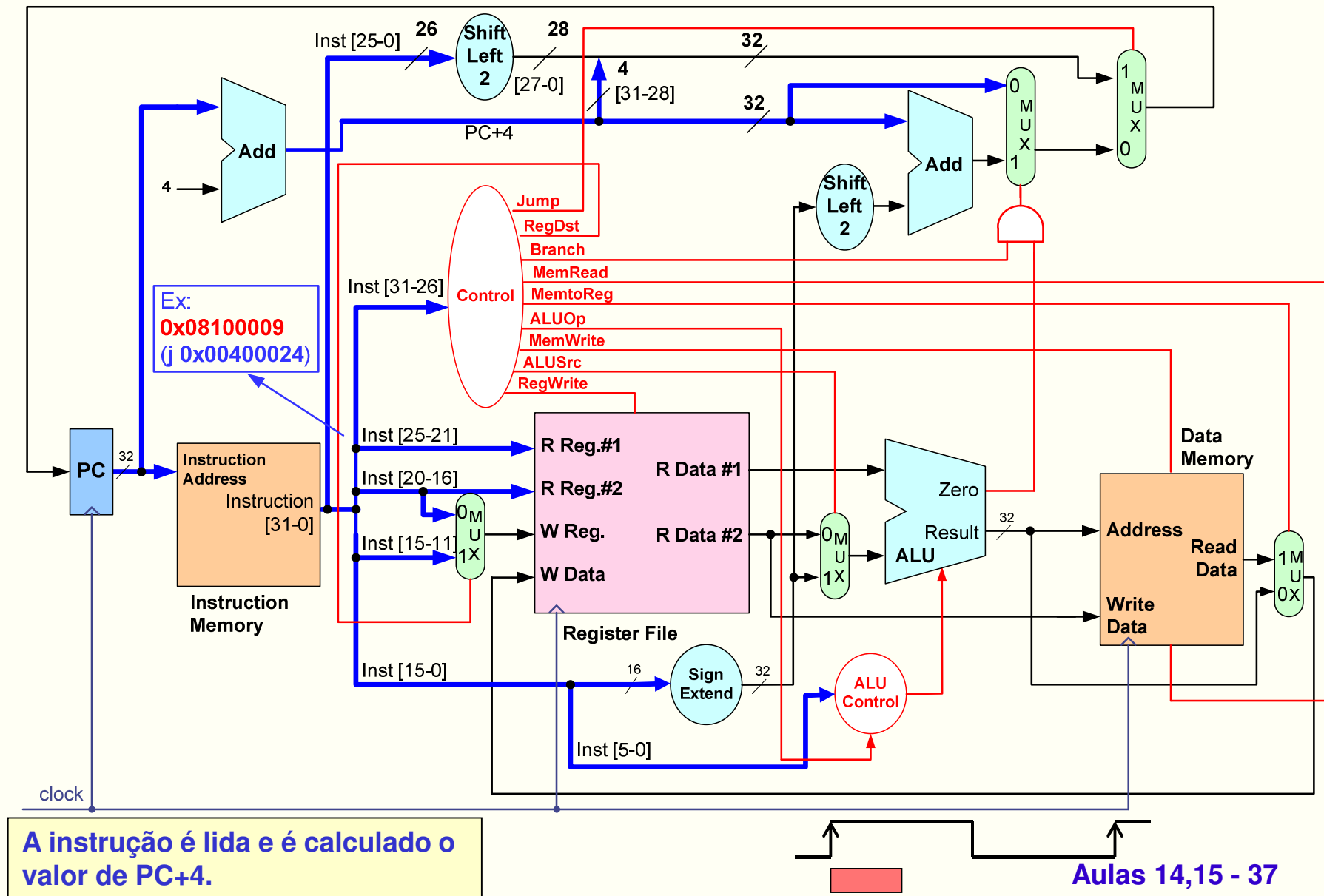
Módulo de atualização do *PC*, completo

```
architecture Behavioral of PCupdate is
    signal s_pc, s_pc4, s_offset : unsigned(31 downto 0);
begin
    s_offset <= unsigned(offset(29 downto 0)) & "00"; -- Left shift
    s_pc4 <= s_pc + 4;
    process(clk)
    begin
        if(rising_edge(clk)) then
            if(reset = '1') then
                s_pc <= (others => '0');
            else
                if(jump = '1') then                -- Jump Target Address
                    s_pc <= s_pc4(31 downto 28) & unsigned(jAddr) & "00";
                elsif(branch = '1' and zero = '1') then
                    s_pc <= s_pc4 + s_offset;      -- Branch Target Address
                else
                    s_pc <= s_pc4;
                end if;
            end if;
        end if;
    end process;
    pc <= std_logic_vector(s_pc);
end Behavioral;
```

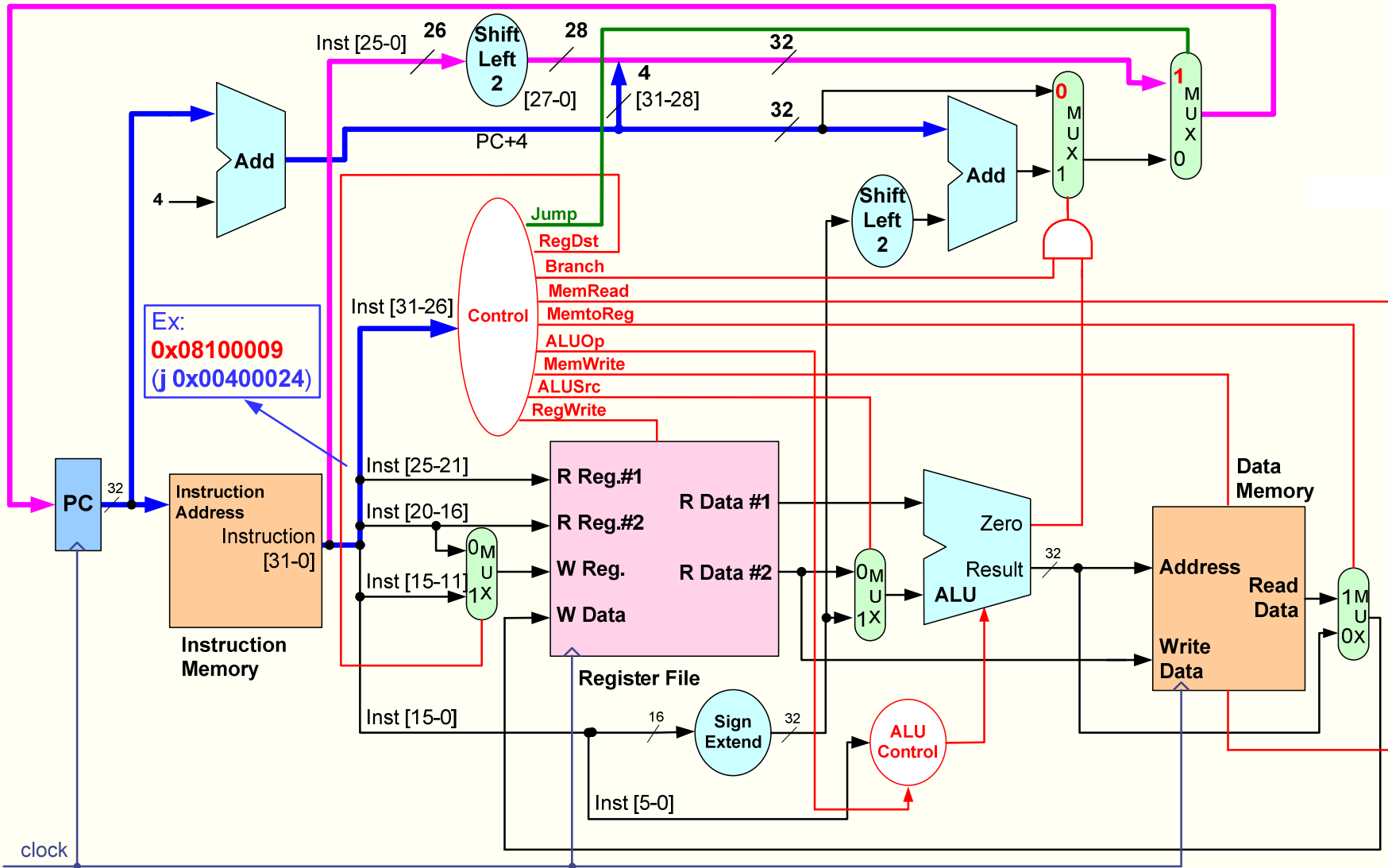
Funcionamento do *datapath* na instrução J

- A instrução é lida e é calculado o valor de PC+4.
- São determinados os sinais de controlo. O novo valor do PC é obtido a partir dos 26 LSB da instrução multiplicados por 4 (*shift left 2*) concatenados com os 4 bits mais significativos do PC atual.

Funcionamento do *datapath* na instrução J (1)

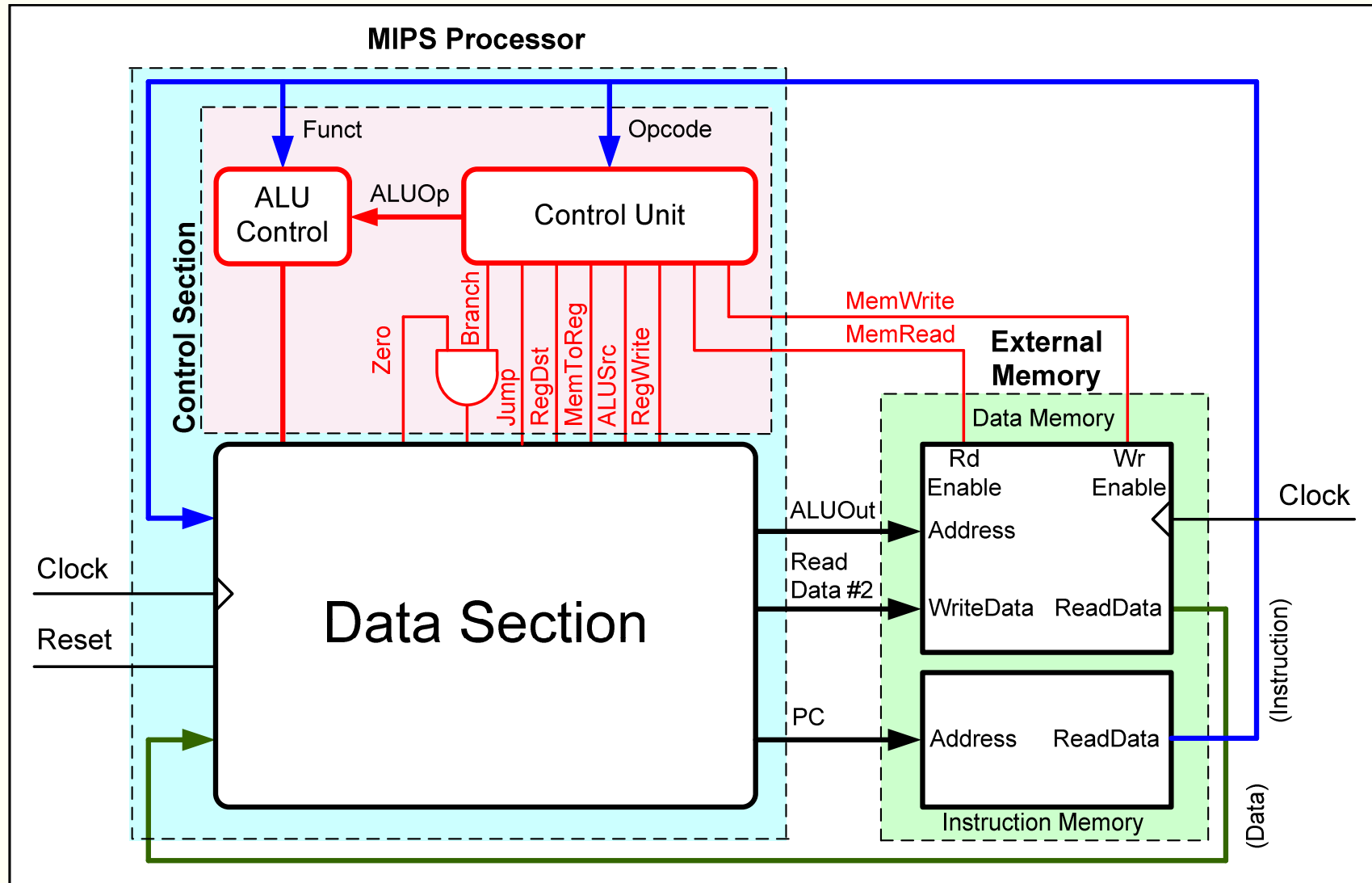


Funcionamento do *datapath* na instrução J (2)



O novo valor do PC é obtido a partir dos 26 LSB da instrução multiplicados por 4 (shift left 2) concatenados com os 4 MSB do PC atual.

Visão global do processador



Execução de uma instrução no *datapath single-cycle* – exemplo

- Vai iniciar-se o *instruction fetch* da instrução apontada pelo registo \$PC (0x00400024). Nesse instante o conteúdo dos registos do CPU e da memória de dados é o indicado. **Qual o conteúdo dos registos após a execução da instrução?**

Endereço	Valor
(...)	(...)
0x10010030	0x63F78395
0x10010034	0xA0FCF3F0
0x10010038	0x147FAF83
(...)	(...)

\$PC	0x00400024
\$3	0x7F421231
\$4	0x15A73C49
\$5	0x10010010



Endereço	Código máquina
(...)	(...)
0x00400020	0x00E82820
0x00400024	0x8CA30024
0x00400028	0x00681824
(...)	(...)

\$PC	0x00400028
\$3	0xA0FCF3F0
\$4	0x15A73C49
\$5	0x10010010

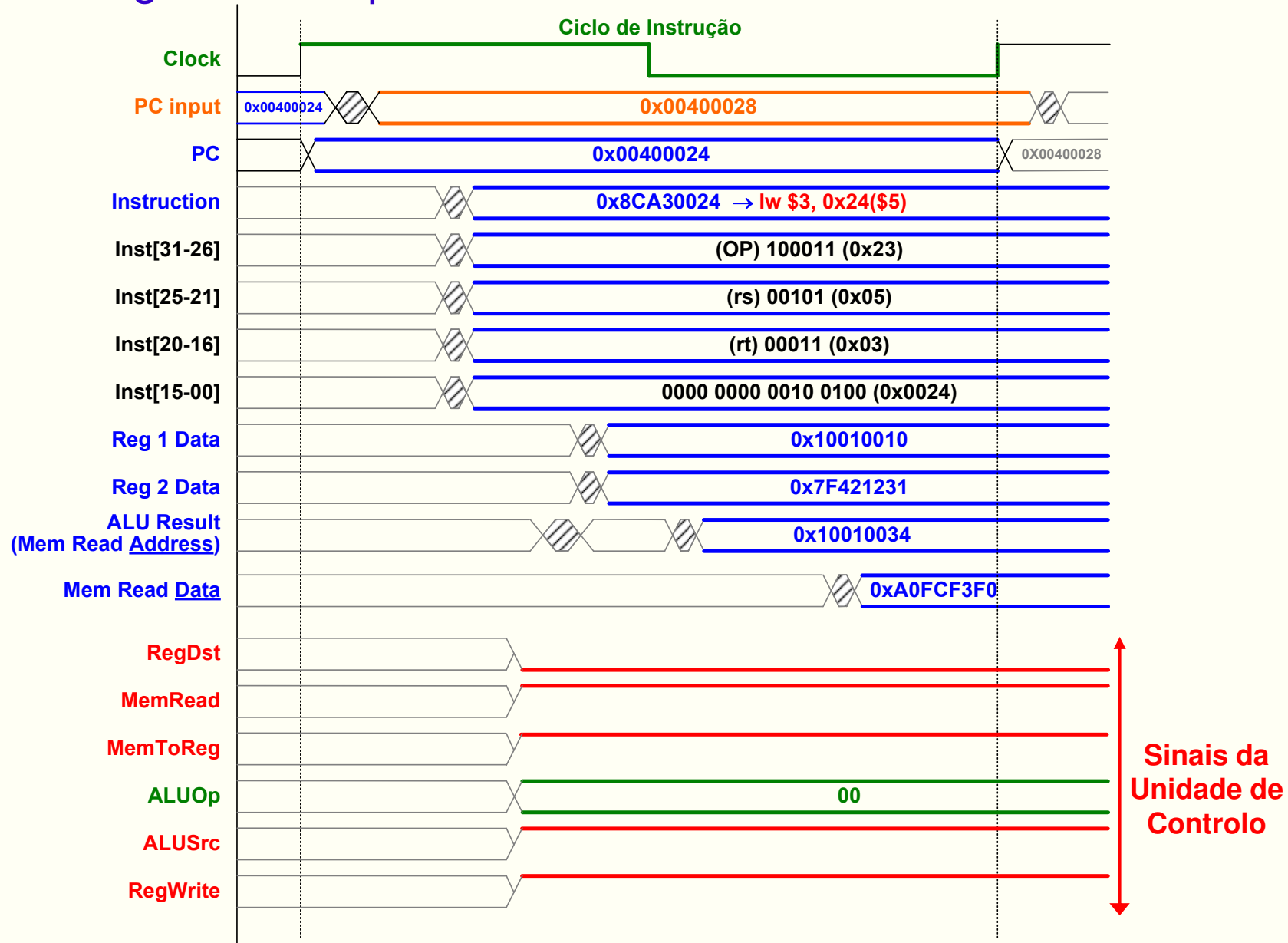
0x8CA30024 → lw \$3, 0x24(\$5)

Mem Addr: 0x10010010 + 0x24 = 0x10010034

1000110010100110000000000100100

\$3 = [0x10010034] = 0xA0FCF3F0

Execução de uma instrução no *datapath single-cycle* – diagrama temporal



op	rs	rt	offset
100011	00101	100011	000000000100100

