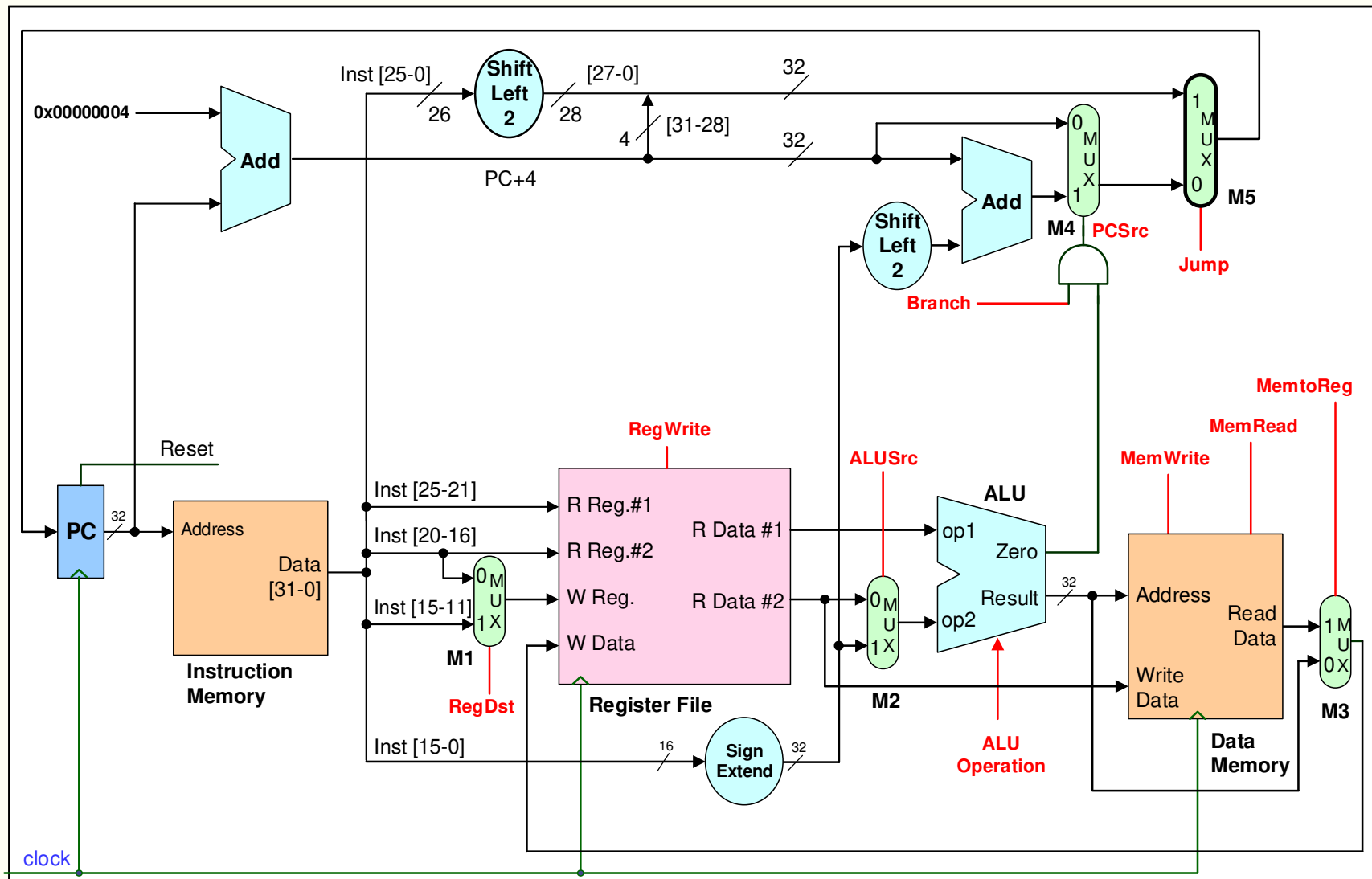


Aulas 17 e 18

- A unidade de controlo principal do *datapath single-cycle*
- A unidade de controlo da ALU
- Implementação das unidades de controlo do *datapath* e da ALU
- Exemplos de funcionamento do *datapath* com unidade de controlo

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

Datapath single-cycle completo



Unidade de controlo

- A unidade de controlo deve gerar os sinais (identificados a vermelho) para:
 - 1) controlar a escrita e/ou a leitura em elementos de estado: **banco de registos** e **memória de dados**
 - 2) definir a operação dos elementos combinatórios: **ALU** e ***multiplexers***
- A operação na ALU é definida com 3 bits (ALU Control):

ALU operation	ALU Control
AND	000
OR	001
ADD	010
SUB	110
SLT	111

Unidade de controlo

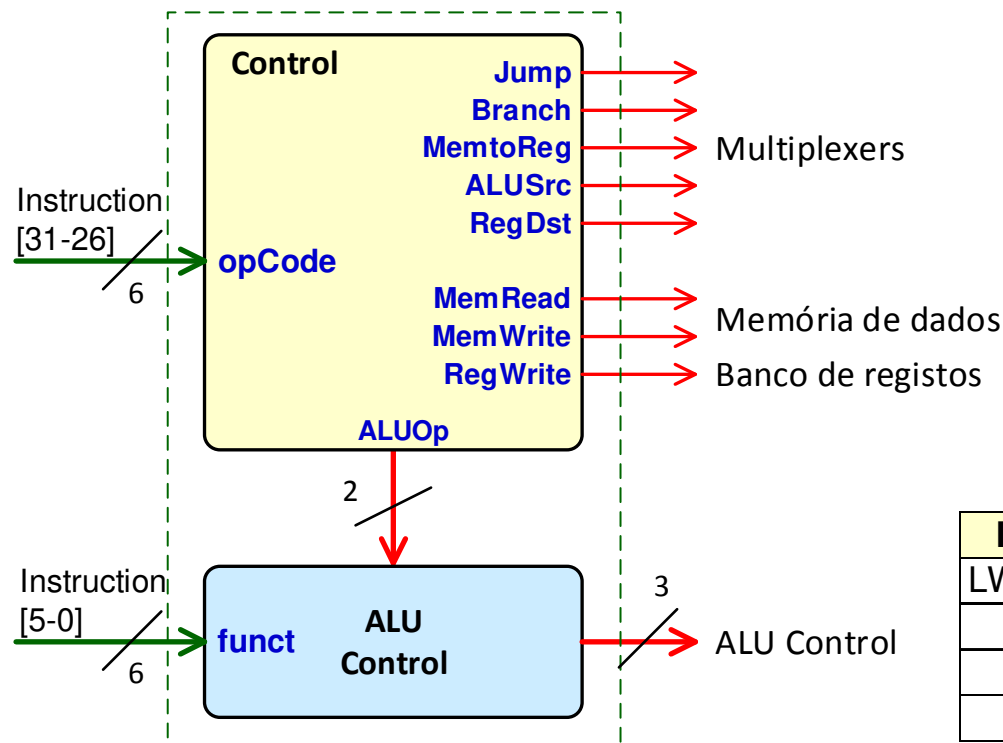
- Alguns dos elementos de estado do *datapath* são acedidos em todos os ciclos de relógio (PC e memória de instruções)
 - Nestes casos não há necessidade de explicitar um sinal de controlo
- Outros elementos de estado podem ser lidos ou escritos dependendo da instrução que estiver a ser executada (memória de dados e banco de registos)
 - Para estes é necessário explicitar os respetivos sinais de controlo
- Nos elementos de estado:
 - a **escrita** é sempre realizada de forma síncrona
 - a **leitura** é sempre realizada de forma assíncrona

Unidade de controlo

- Todas as instruções (exceto o "j") usam a ALU:
 - **LW e SW** – para calcular o endereço da memória externa (soma)
 - **Branch if equal / not equal** – para determinar se os operandos são iguais ou diferentes (subtração)
 - **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 (opcode=0):
 $ALUControl = f(opcode, funct)$
 - do campo **opcode** nas restantes instruções:
 $ALUControl = f(opcode)$

Unidade de controlo

- A unidade de controlo pode ser sub-dividida em duas: 1) controlo de multiplexers e elementos de estado; 2) controlo da ALU

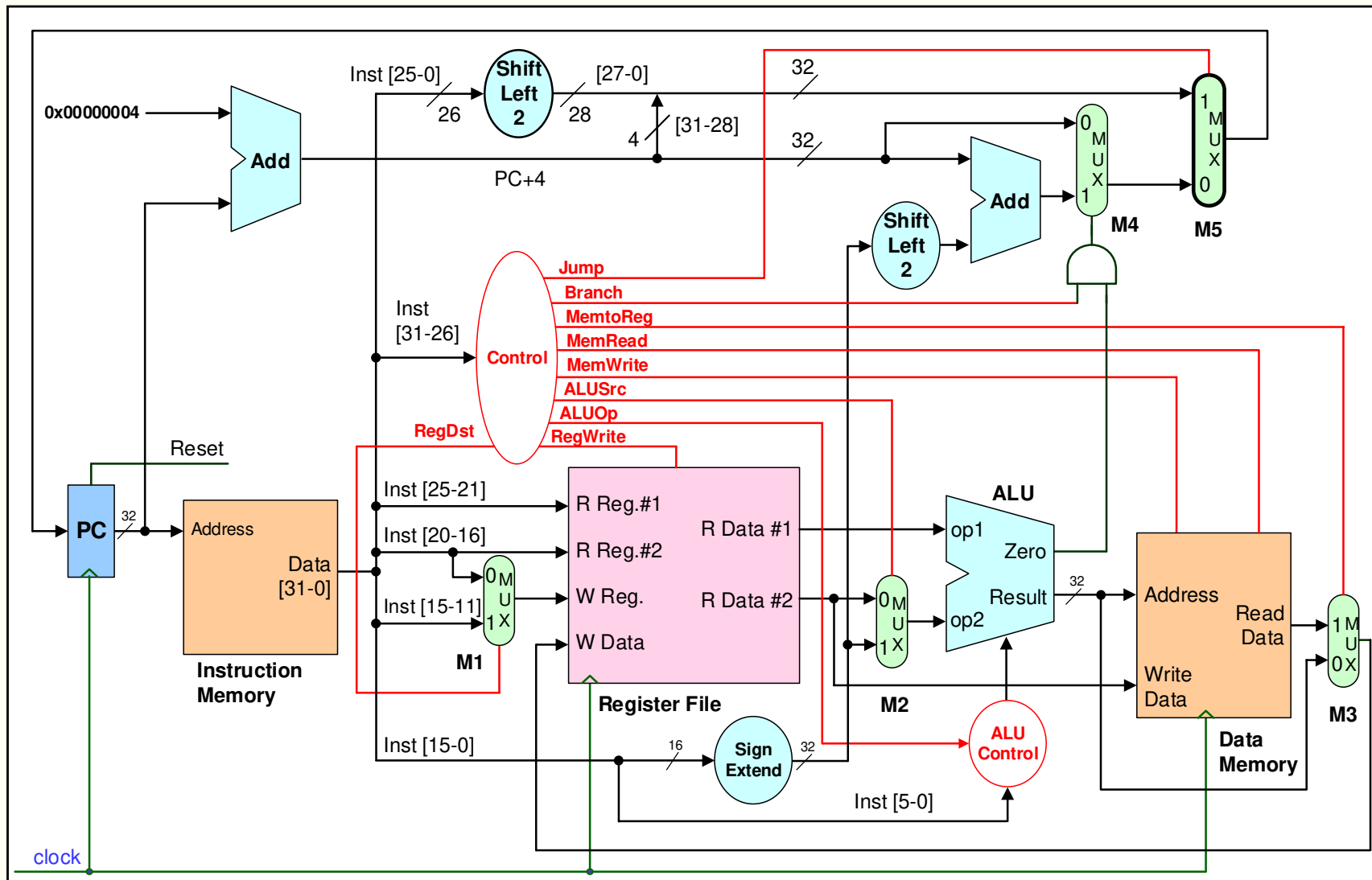


ALU Control	ALU operation
000	AND
001	OR
010	ADD
110	SUB
111	SLT

Instruction	ALUOp	ALU operation
LW, SW, ADDI	00	ADD
BEQ	01	SUB
R-Type	10	Depends on "funct"
SLTI	11	SLT

- A operação da ALU é definida em conjunto com a unidade de controlo principal, em função dos campos "opcode" e "funct"

Datapath single-cycle com unidade de controlo



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 tabela seguinte

ALU Control	ALU operation
000	AND
001	OR
010	ADD
110	SUB
111	SLT

Instruction	opcode	funct	ALU Operation	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
jump	000010 ("j")	xxxxxx	-	xx	xxx

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"; -- ADD
      when "01" => -- BEQ
        ALUcontrol <= "110"; -- SUB
      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 <= "---";
        end case;
      when "11" => -- SLTI
        ALUcontrol <= "111";
    end case;
  end process;
end Behavioral;

```

ALU Control	ALU operation
0 0 0	AND
0 0 1	OR
0 1 0	ADD
1 1 0	SUB
1 1 1	SLT

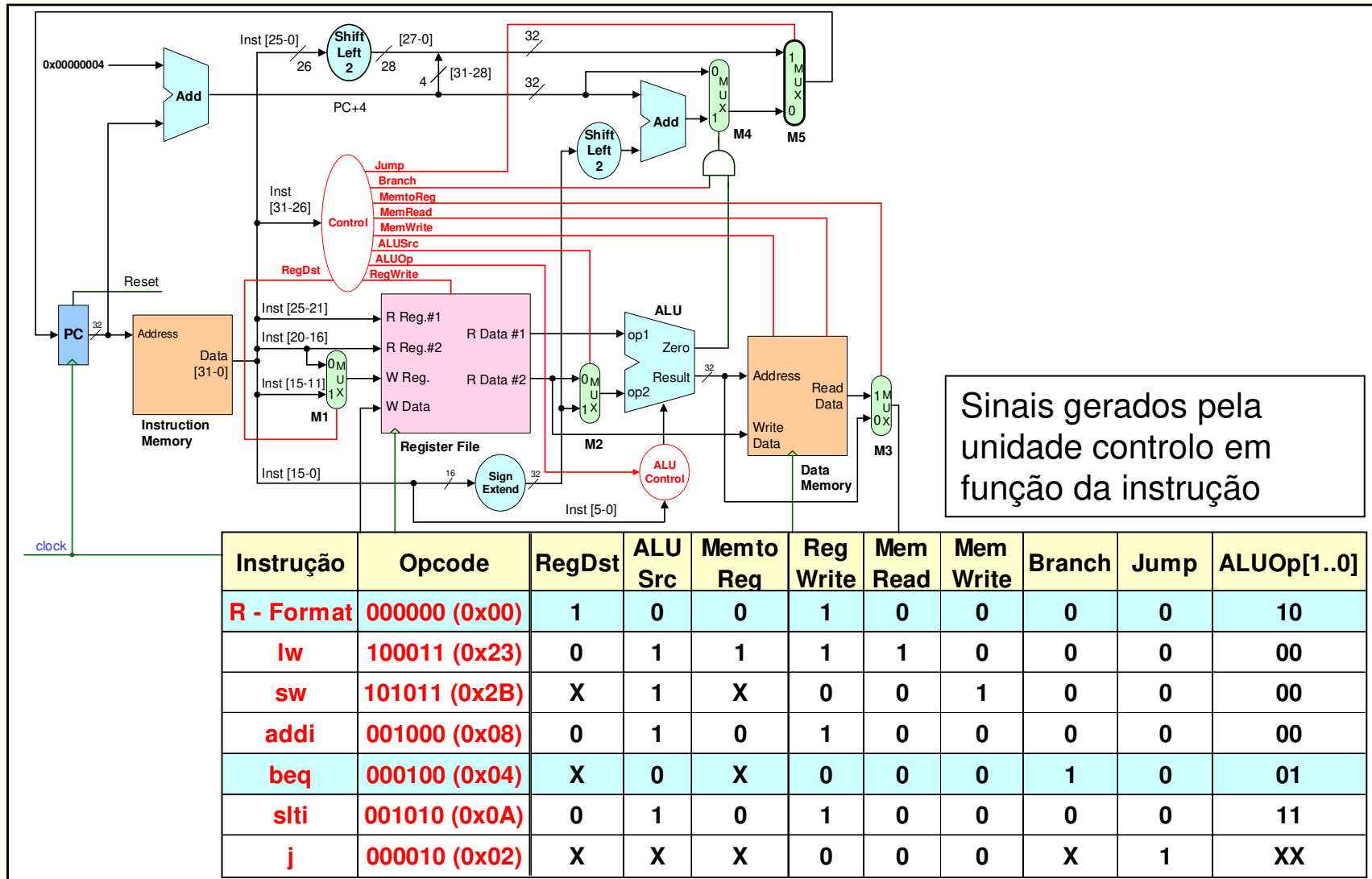
Instruction	ALUOp	ALU operation
LW, SW, ADDI	00	ADD
BEQ	01	SUB
R-Type	10	Depends on "funct"
SLTI	11	SLT

Unidade de controlo principal

- É necessário especificar um total de oito sinais de controlo (para além do ALUOp):

Sinal	Efeito quando não ativo ('0')	Efeito quando ativo ('1')
MemRead	Nenhum (barramento de dados da memória em alta impedância)	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
RegWrite	Nenhum	O registo indicado no endereço de escrita é alterado pelo valor presente na entrada de dados
RegDst	O endereço do registo destino provém do campo "rt"	O endereço do registo destino provém do campo "rd"
ALUSrc	O segundo operando da ALU provém da segunda saída do <i>Register File</i>	O segundo operando da ALU provém dos 16 bits menos significativos da instrução após extensão do sinal
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
Branch	Nenhum	Indica que a instrução é um branch condicional
PCSrc	O PC é substituído pelo seu valor actual mais 4	O PC é substituído pelo resultado do somador que calcula o endereço alvo do <i>branch</i> condicional
Jump	Nenhum	Indica que a instrução é um <i>jump</i> incondicional

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;
        Jump        : 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'; Jump  <= '0';
```

```
    ALUOp    <= "00";
```

```
    case OpCode is
```

```
      when "000000" =>  -- R-Type instructions
```

```
        ALUOp  <= "10"; RegDst <= '1'; RegWrite <= '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 "000100" =>  -- BEQ
```

```
        ALUOp  <= "01"; Branch <= '1';
```

```
      when "001010" =>  -- SLTI
```

```
        ALUOp  <= "11"; ALUSrc <= '1'; RegWrite <= '1';
```

```
      when "000010" =>  -- J
```

```
        Jump   <= '1';
```

```
      when others =>
```

```
        end case;
```

```
    end process;
```

```
end Behavioral;
```

Instrução	Opcode	RegDst	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALUOp[1..0]
R - Format	000000 (0x00)	1	0	0	1	0	0	0	0	10
lw	100011 (0x23)	0	1	1	1	1	0	0	0	00
sw	101011 (0x2B)	X	1	X	0	0	1	0	0	00
addi	001000 (0x08)	0	1	0	1	0	0	0	0	00
beq	000100 (0x04)	X	0	X	0	0	0	1	0	01
slti	001010 (0x0A)	0	1	0	1	0	0	0	0	11
j	000010 (0x02)	X	X	X	0	0	0	X	1	XX

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, considerar que a utilização dos vários elementos operativos ocorre em sequência e decorre ao longo de um conjunto de operações
- A sequência de operações culmina 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

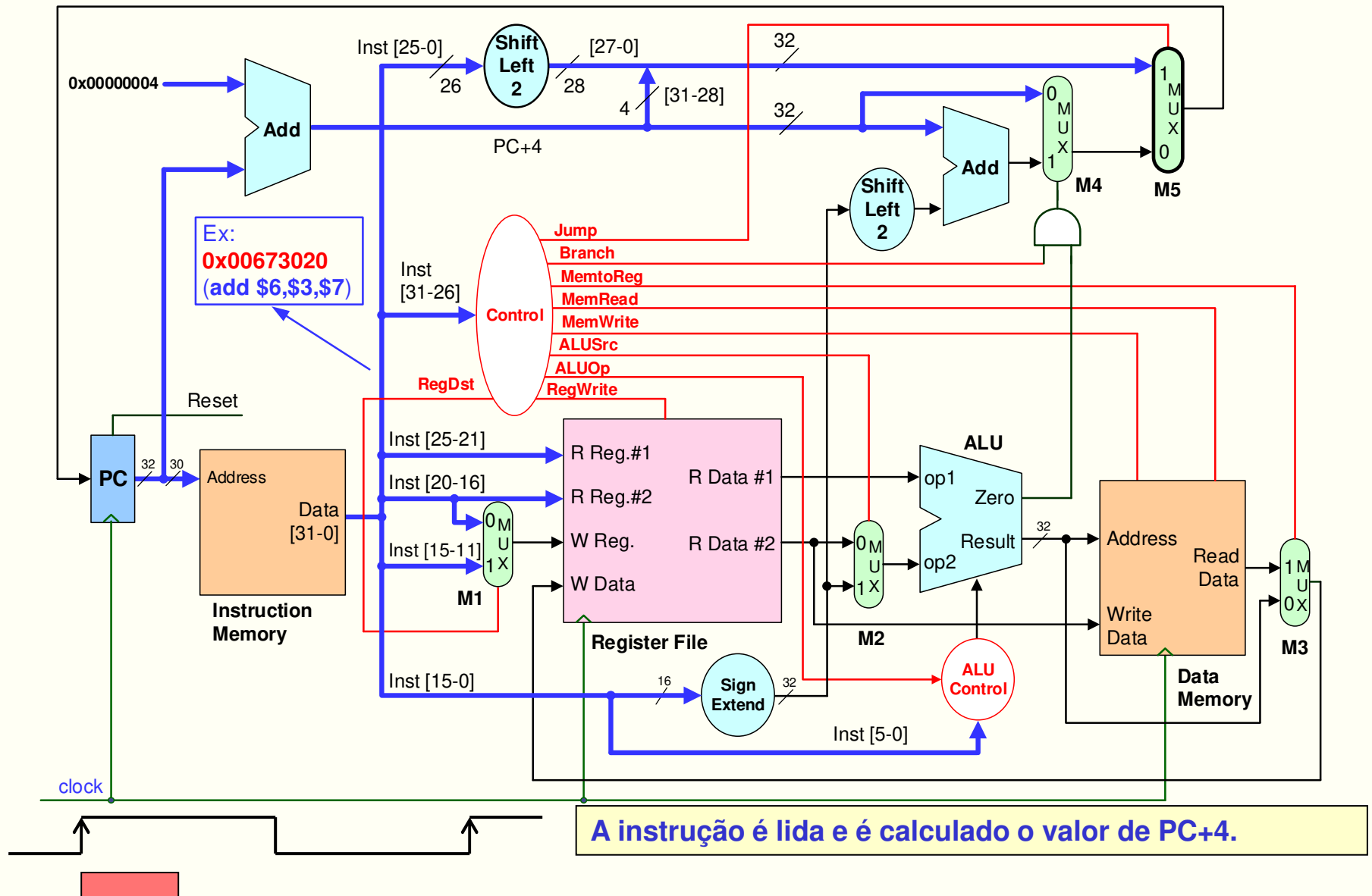
Análise do funcionamento do *datapath* – operações

- *Fetch* de uma instrução e cálculo do endereço da próxima instrução
- Leitura de dois registros do Banco de Registos
- A ALU opera sobre dois valores (a origem do segundo operando 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**): BTA ou PC+4

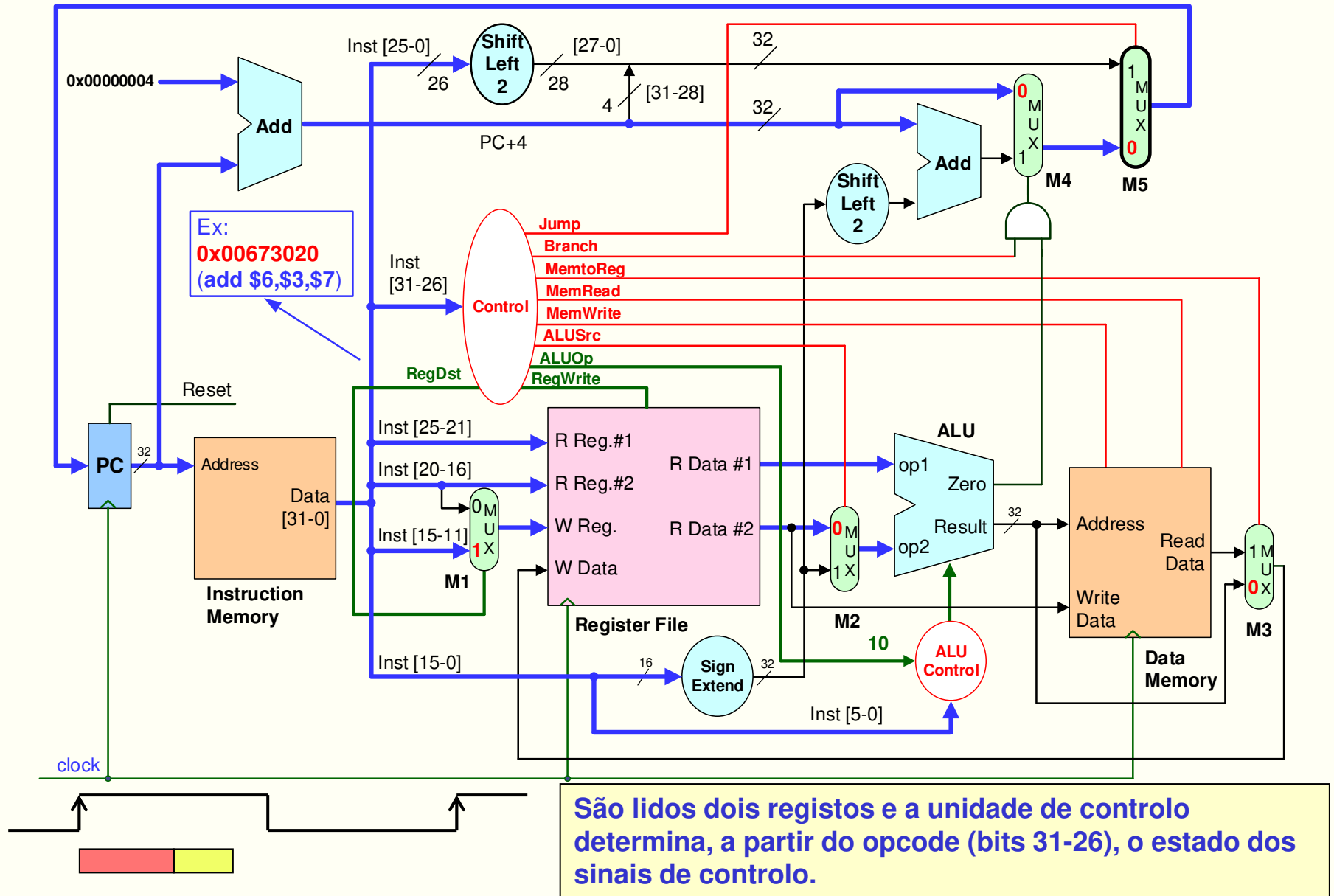
Funcionamento do *datapath* nas instruções tipo R

- 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 no campo *funct* (**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

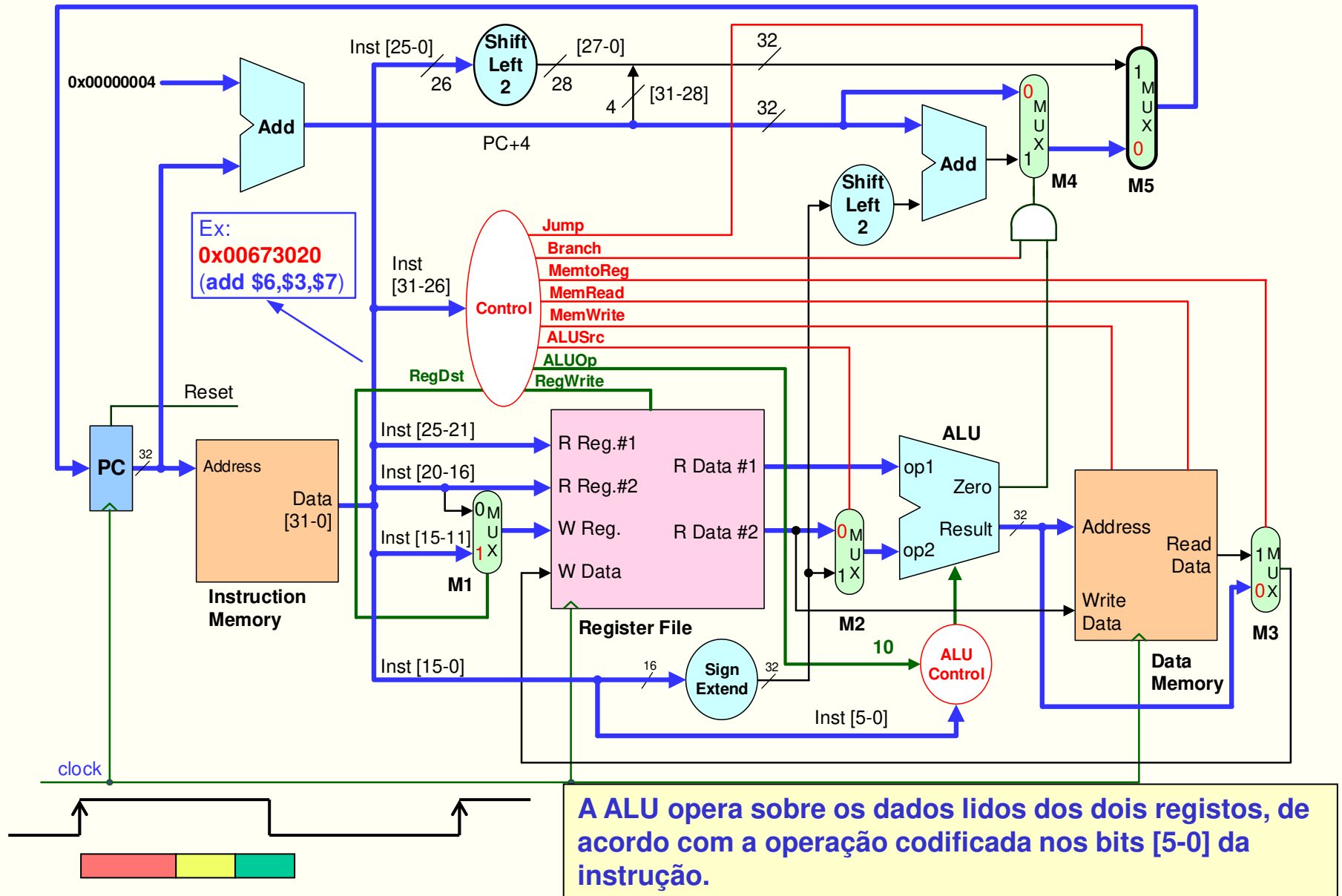
Funcionamento do *datapath* nas instruções tipo R (1)



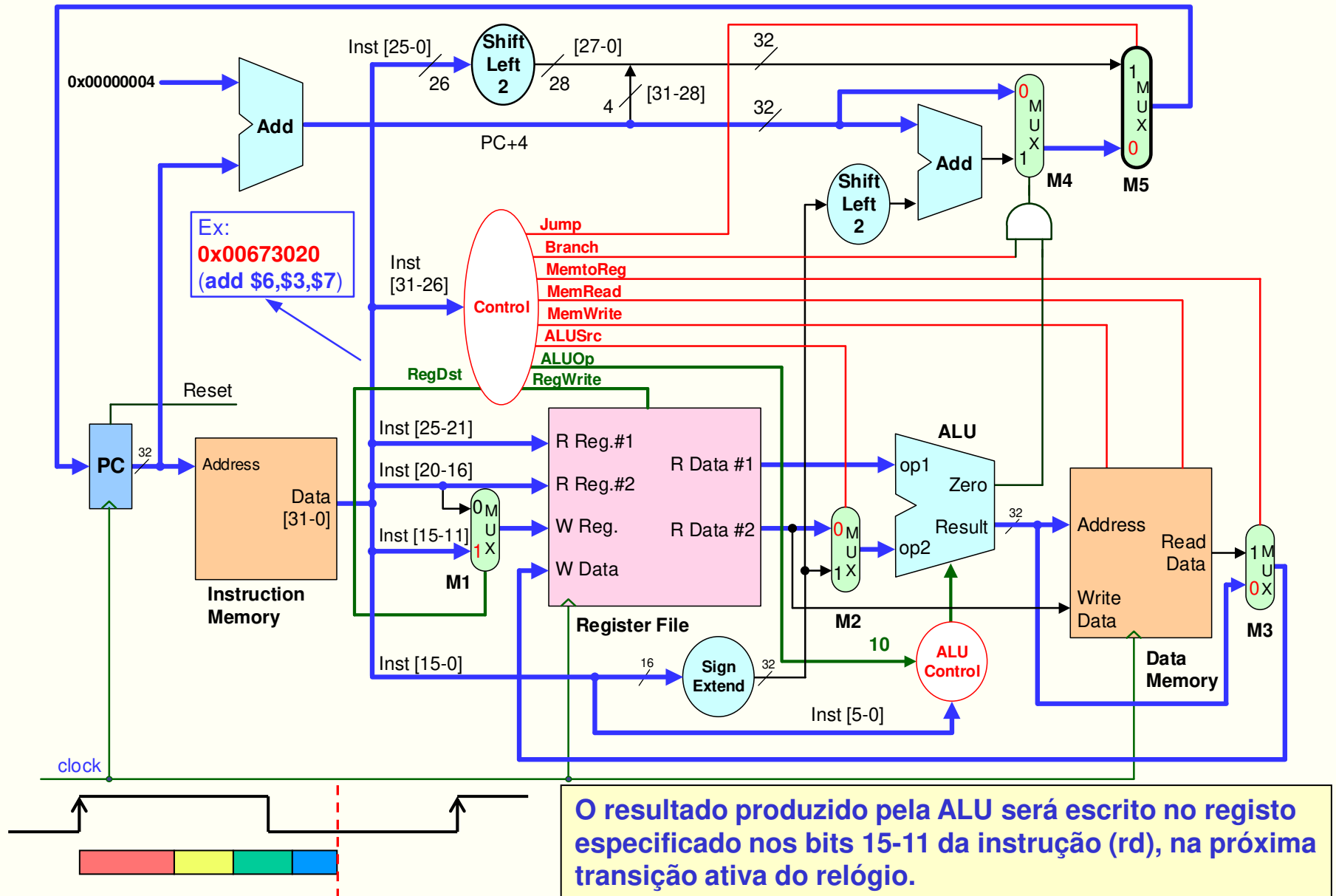
Funcionamento do *datapath* nas instruções tipo R (2)



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



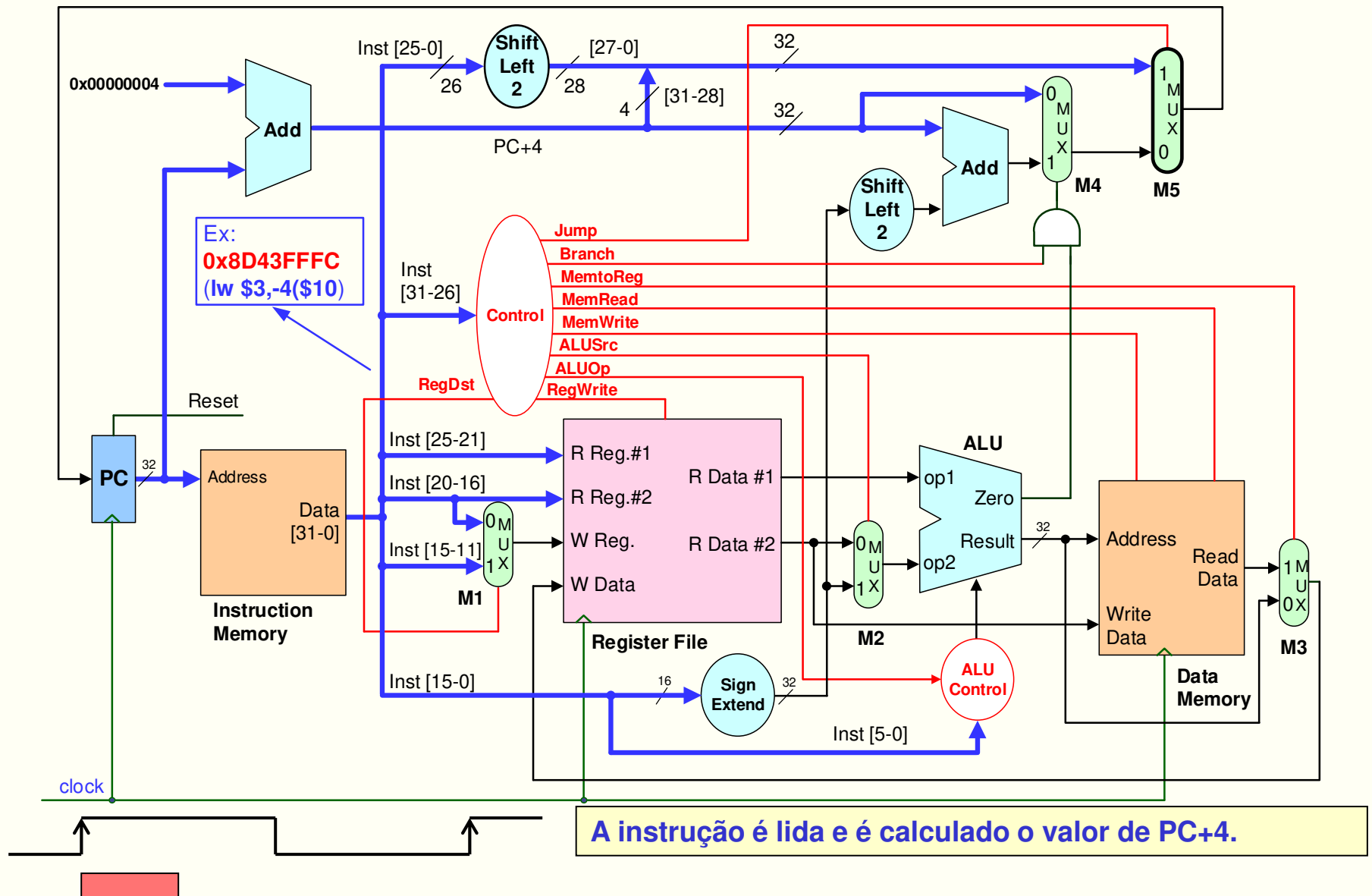
Funcionamento do *datapath* nas instruções tipo R (4)



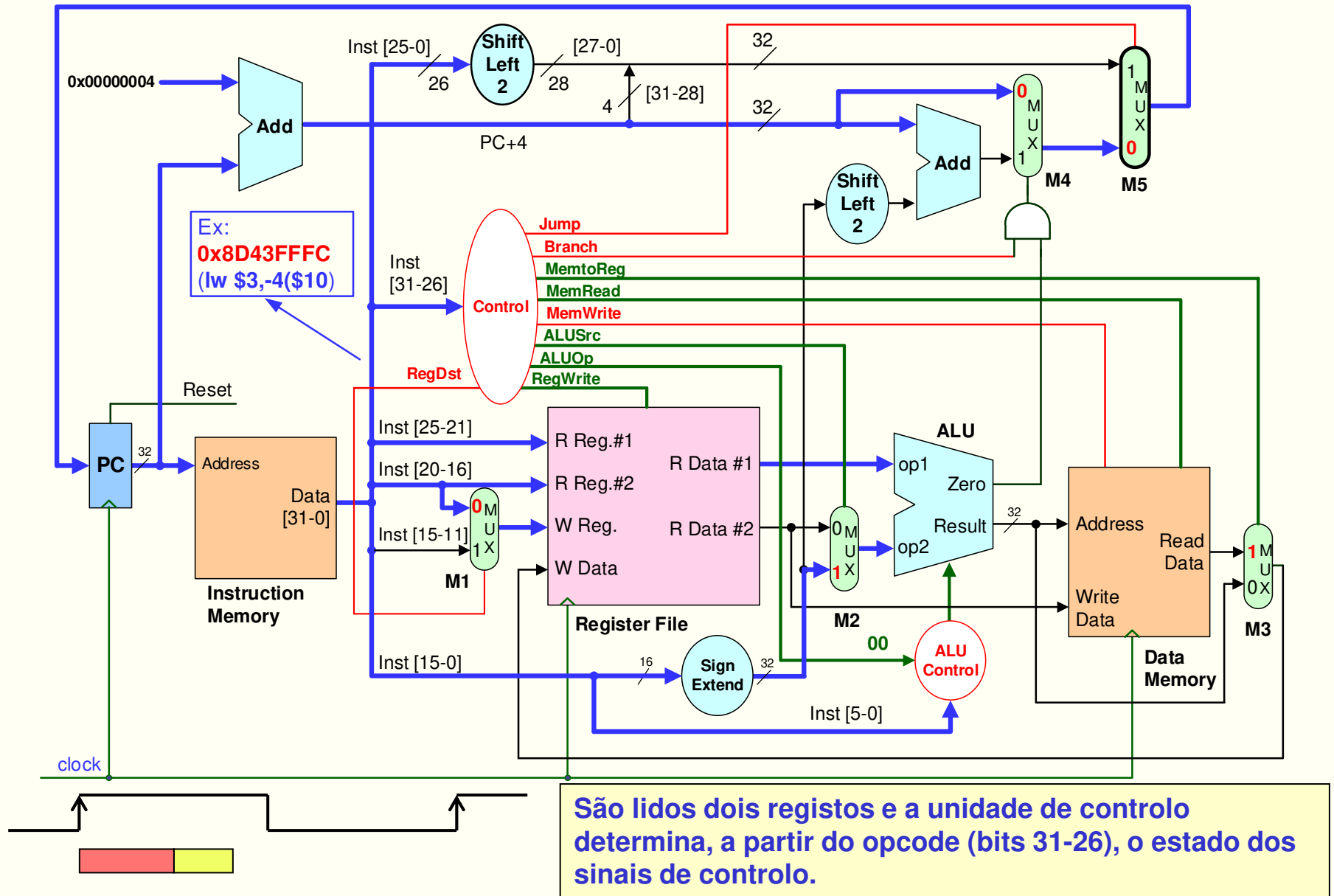
Funcionamento do *datapath* na instrução LW

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

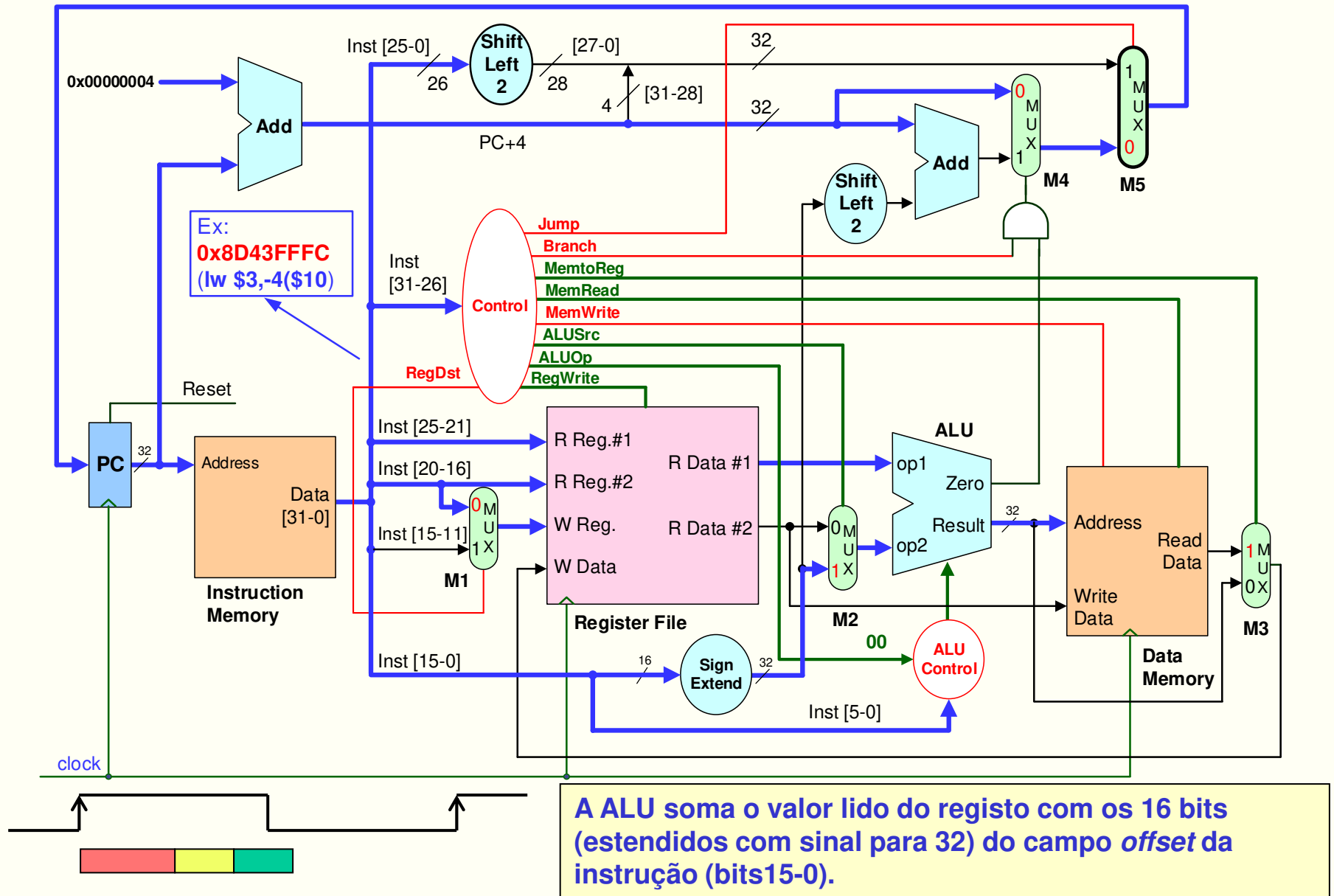
Funcionamento do *datapath* na instrução LW (1)



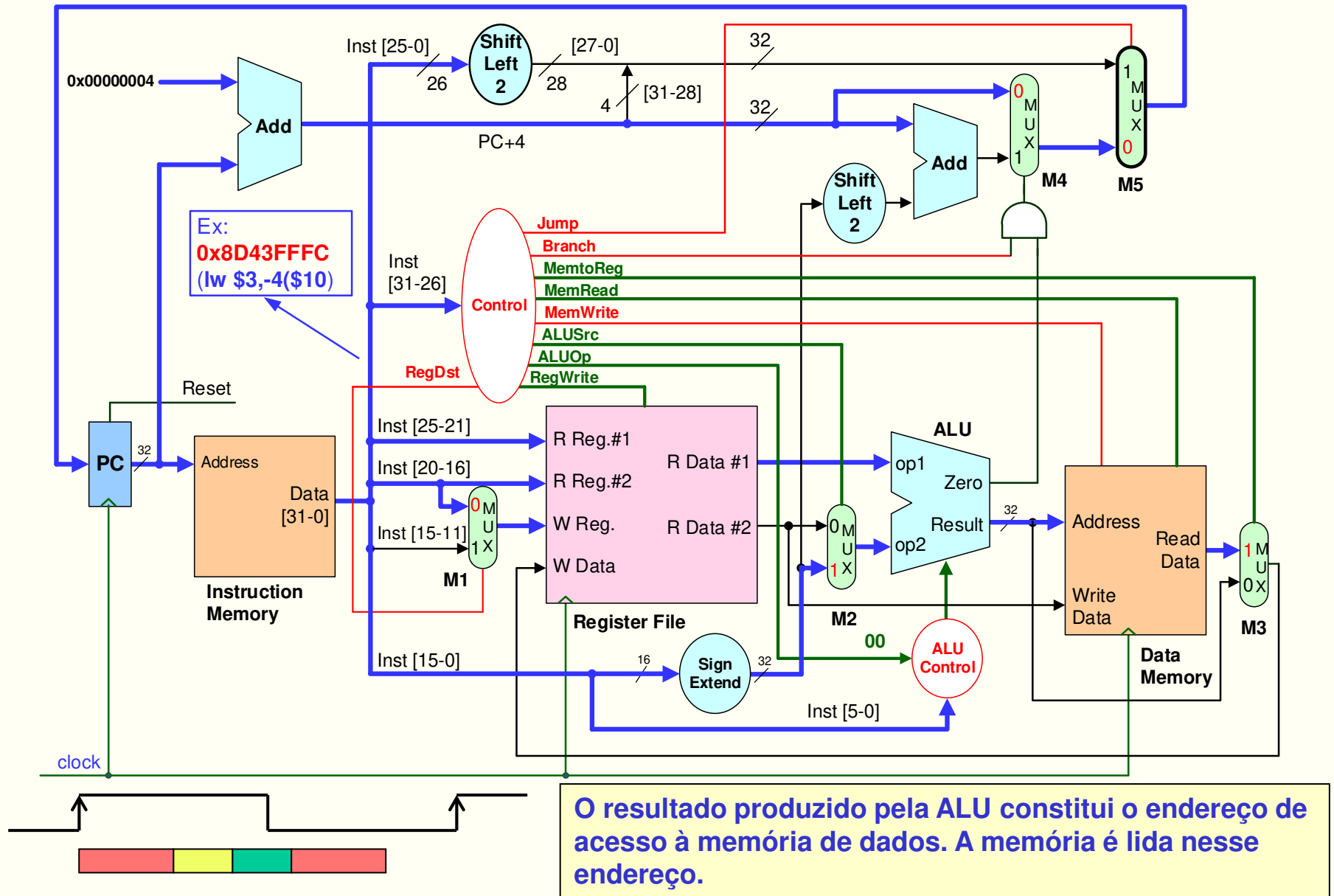
Funcionamento do *datapath* na instrução LW (2)



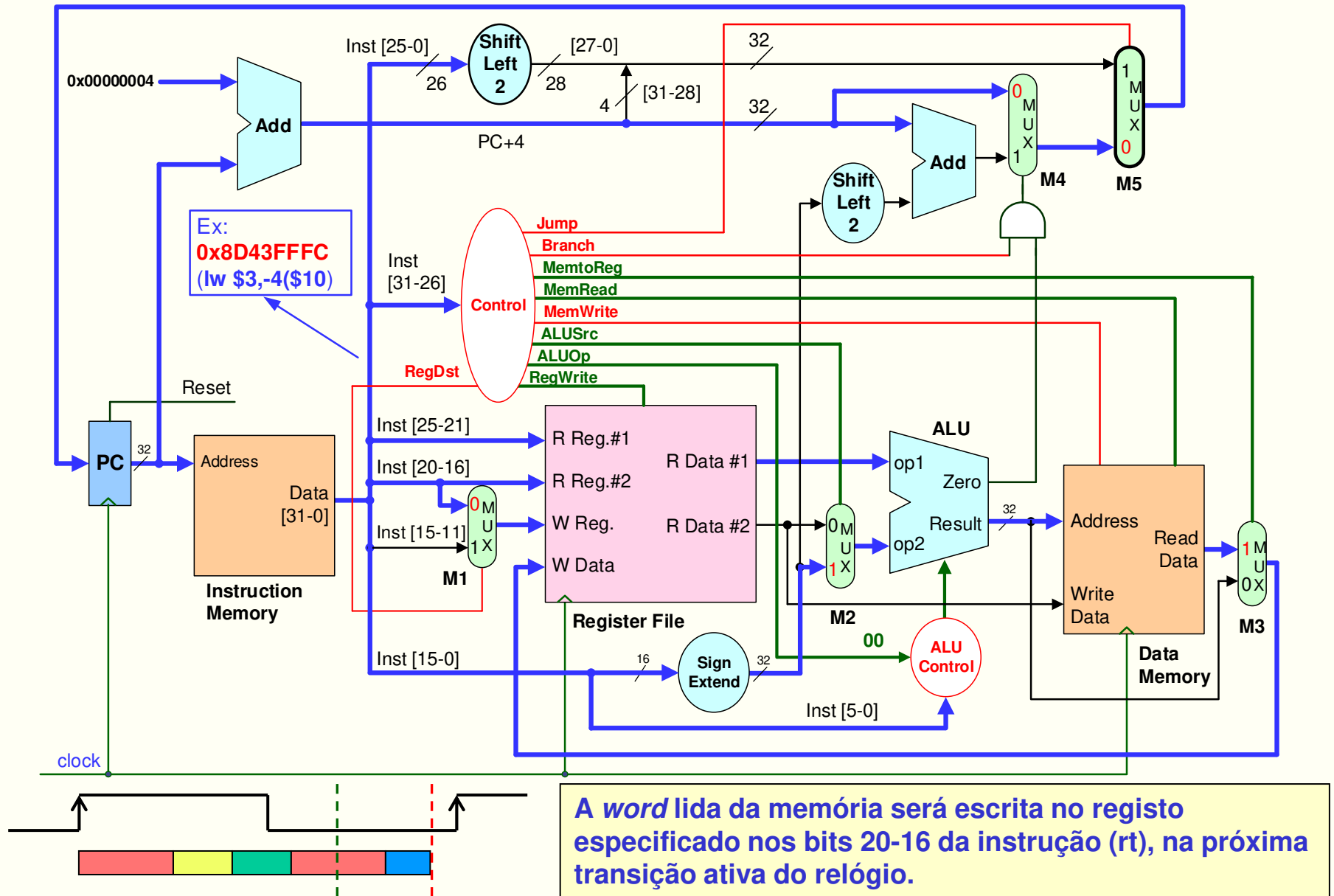
Funcionamento do *datapath* na instrução LW (3)



Funcionamento do *datapath* na instrução LW (4)



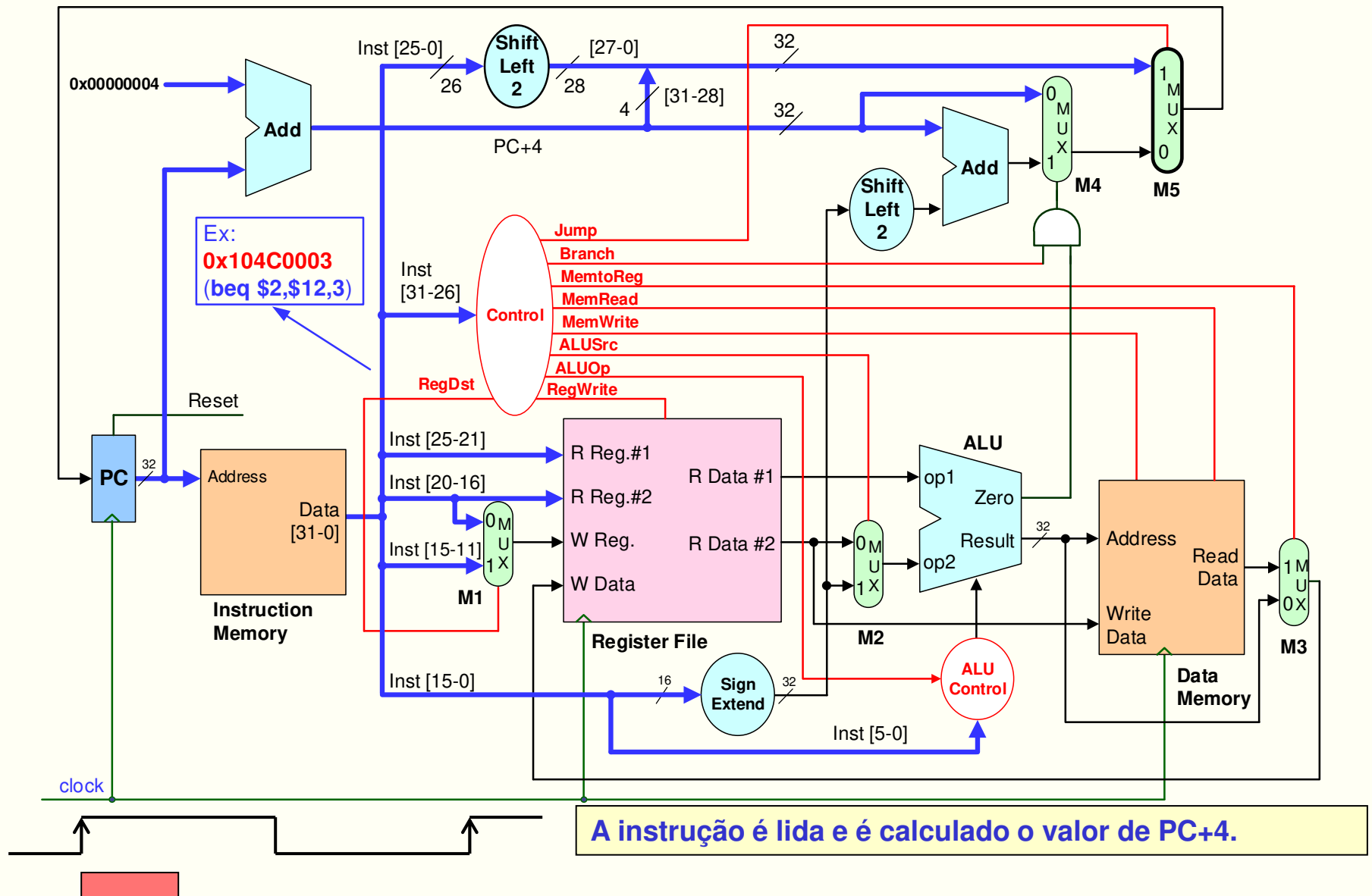
Funcionamento do *datapath* na instrução LW (5)



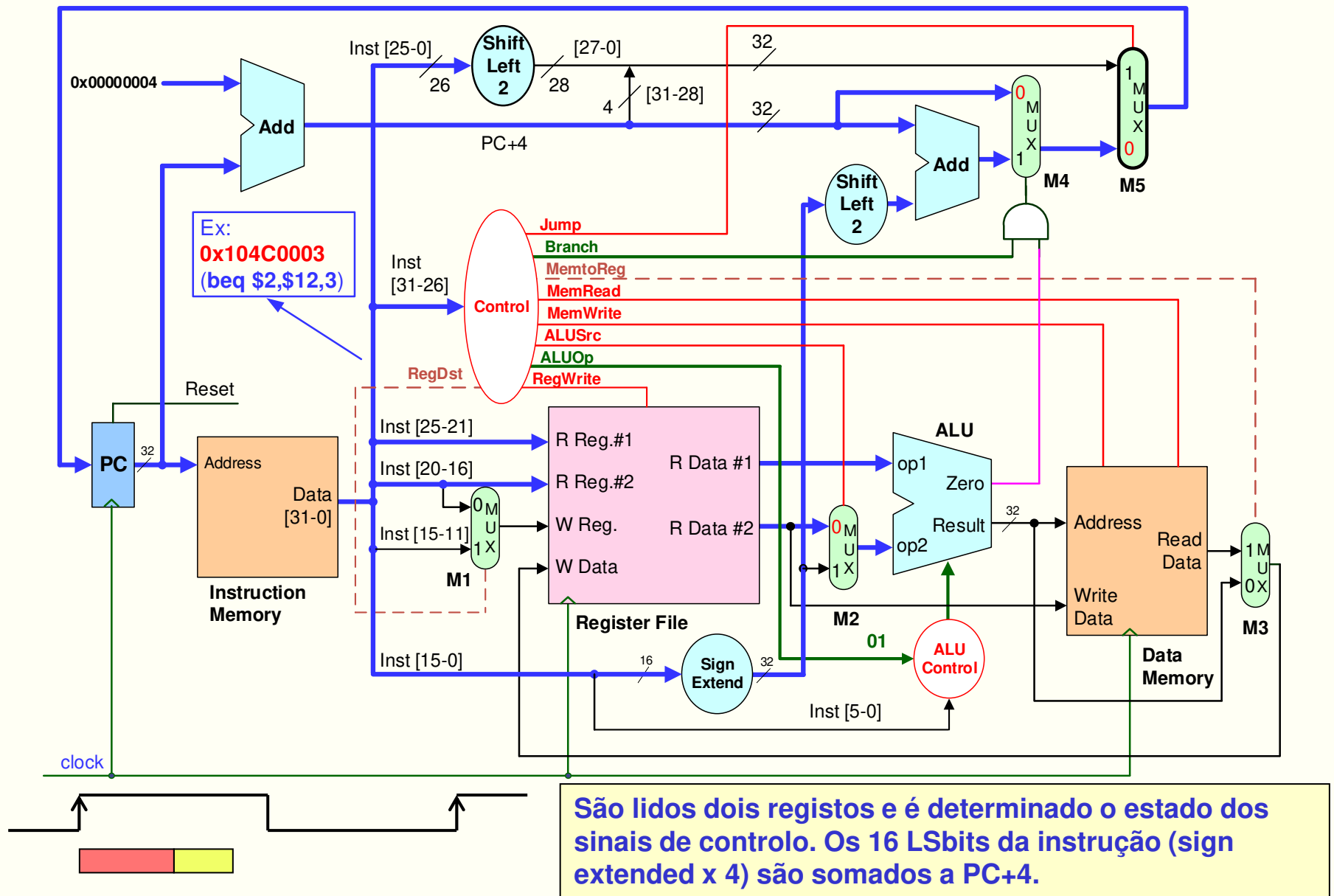
Funcionamento do *datapath* na instrução BEQ

- 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 16 LSbits 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

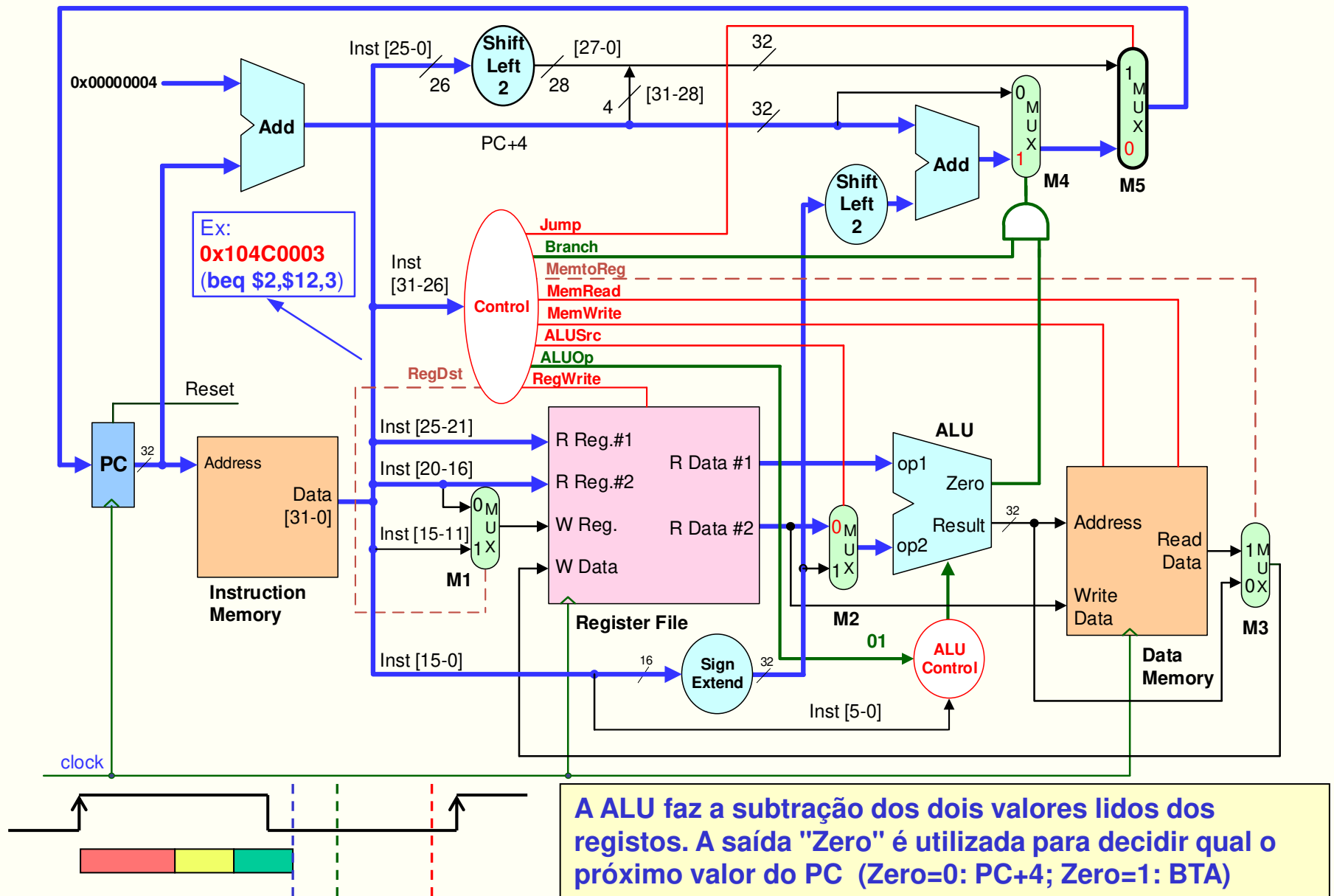
Funcionamento do *datapath* na instrução BEQ (1)



Funcionamento do *datapath* na instrução BEQ (2)



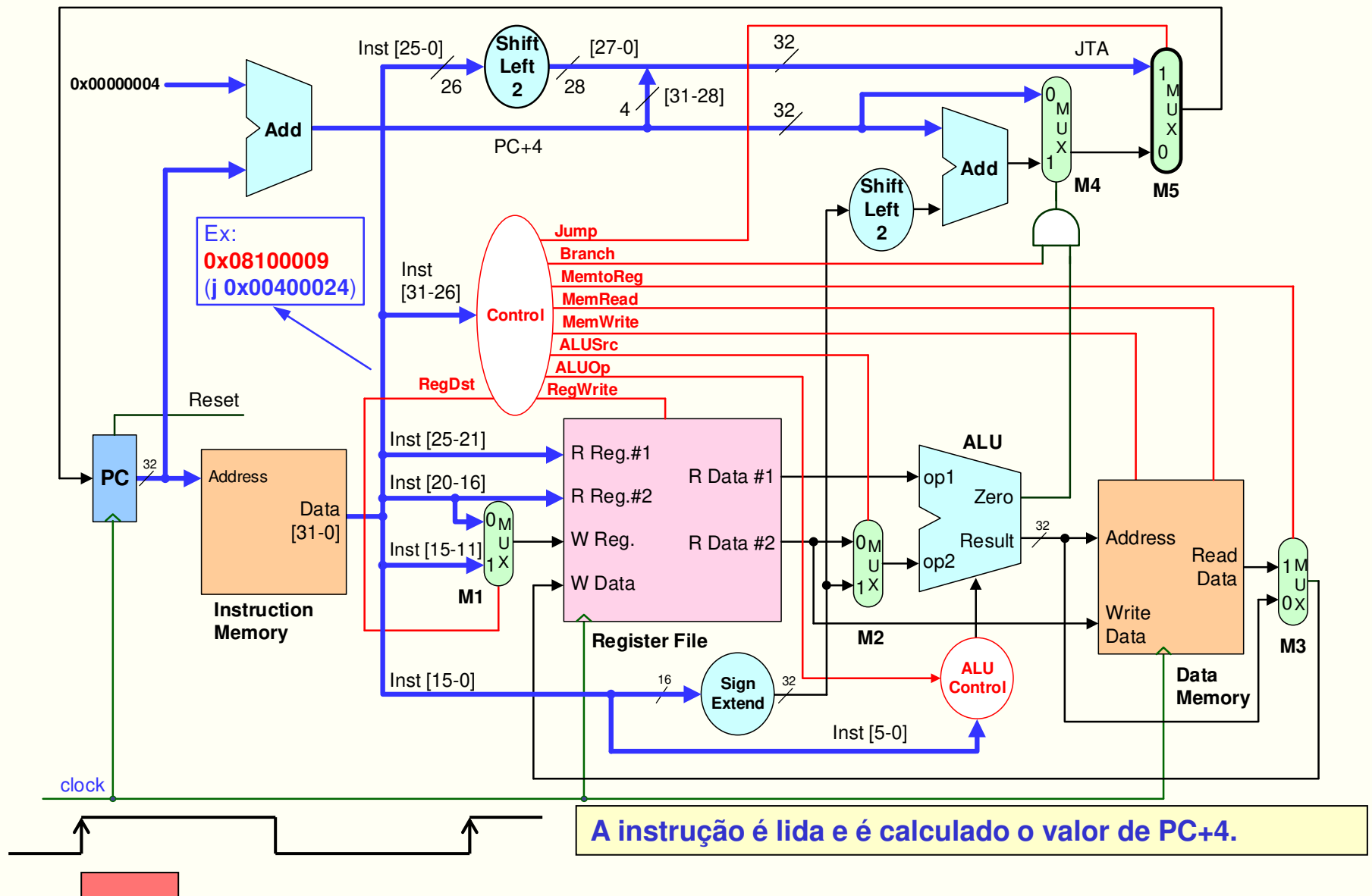
Funcionamento do *datapath* na instrução BEQ (3)



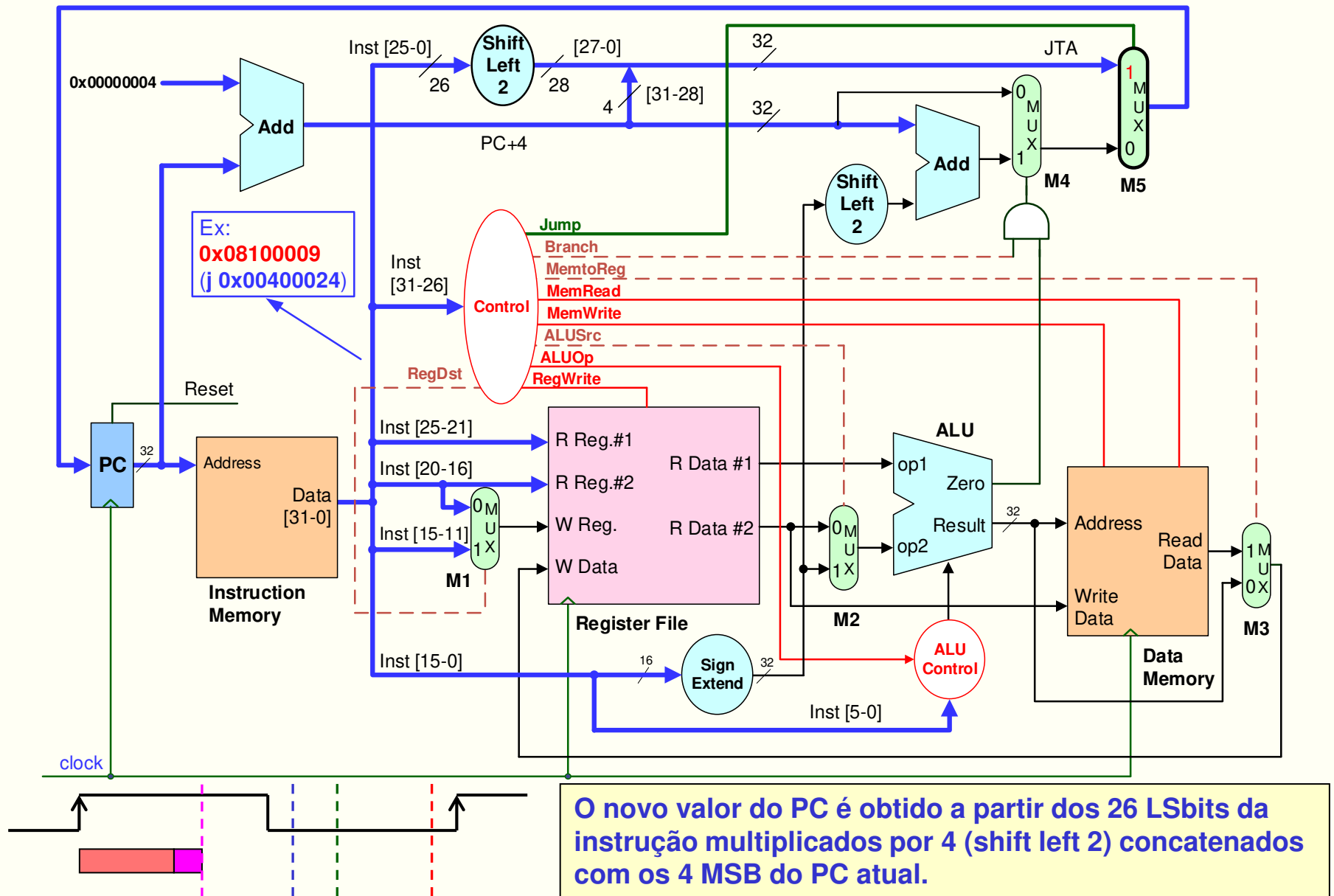
Funcionamento do *datapath* na instrução J

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

Funcionamento do *datapath* na instrução J (1)



Funcionamento do *datapath* na instrução J (2)



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

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

Memória de dados	Endereço	Valor	Memória de instruções	Endereço	Código máquina
	(...)	(...)		(...)	(...)
	0x10010030	0x63F78395		0x00400020	0x00E82820
	0x10010034	0xA0FCF3F0		0x00400024	0x8CA30024
	0x10010038	0x147FAF83		0x00400028	0x00681824
CPU antes	(...)	(...)	CPU depois	(...)	(...)
	PC	0x00400024		PC	0x00400028
	\$3	0x7F421231		\$3	0xA0FCF3F0
	\$4	0x15A73C49		\$4	0x15A73C49
	\$5	0x10010010		\$5	0x10010010

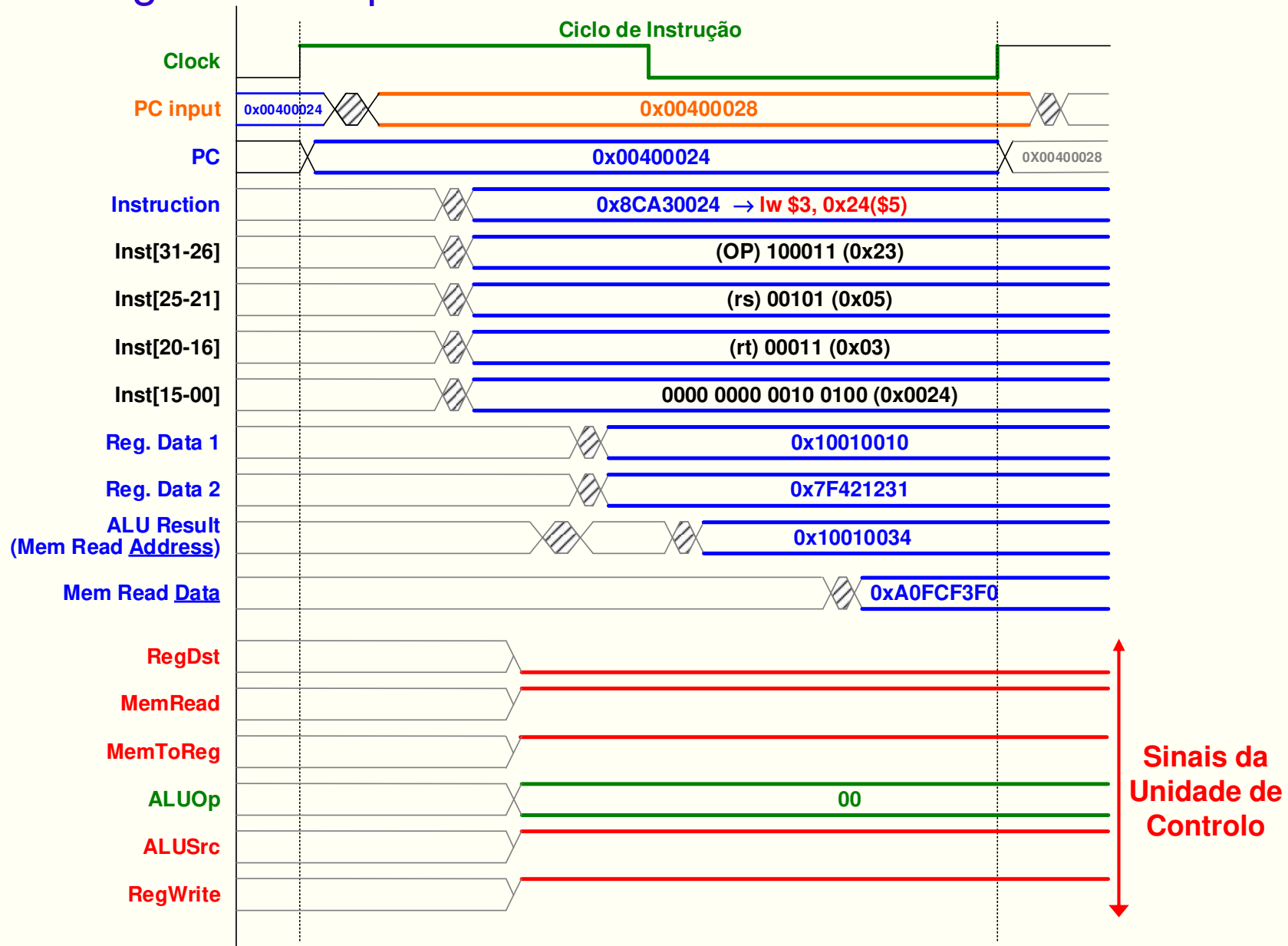
0x8CA30024 → lw \$3, 0x24(\$5)

Mem Addr: 0x10010010 + 0x24 = 0x10010034

100011**00101000**11000000000100100

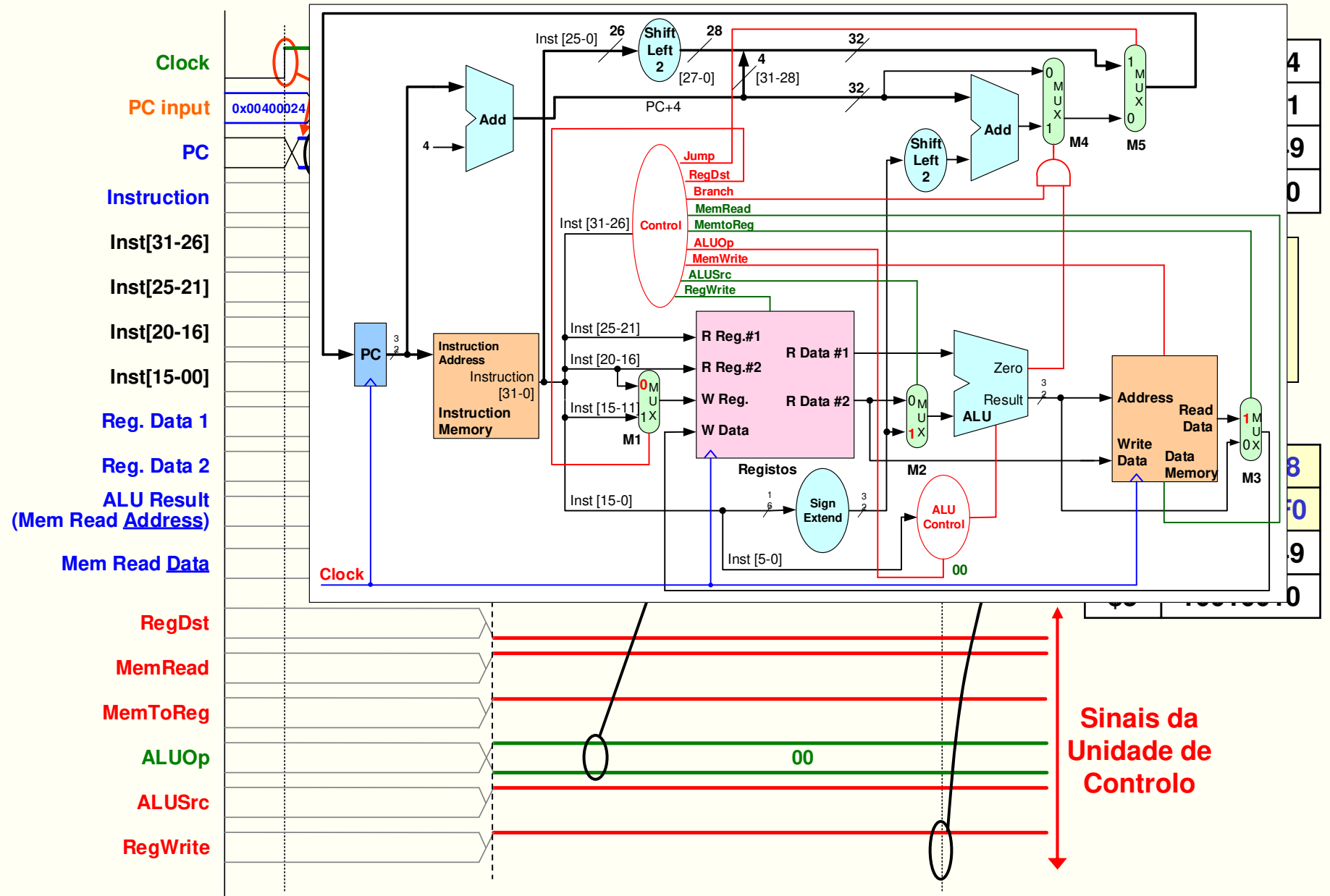
\$3 = [0x10010034] = 0xA0FCF3F0

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



0x00400024	0x8CA30024
0x10010034	0xA0FCF3F0

op	rs	rt	offset
100011	00101	100011	0000000000100100



op

rs

rt

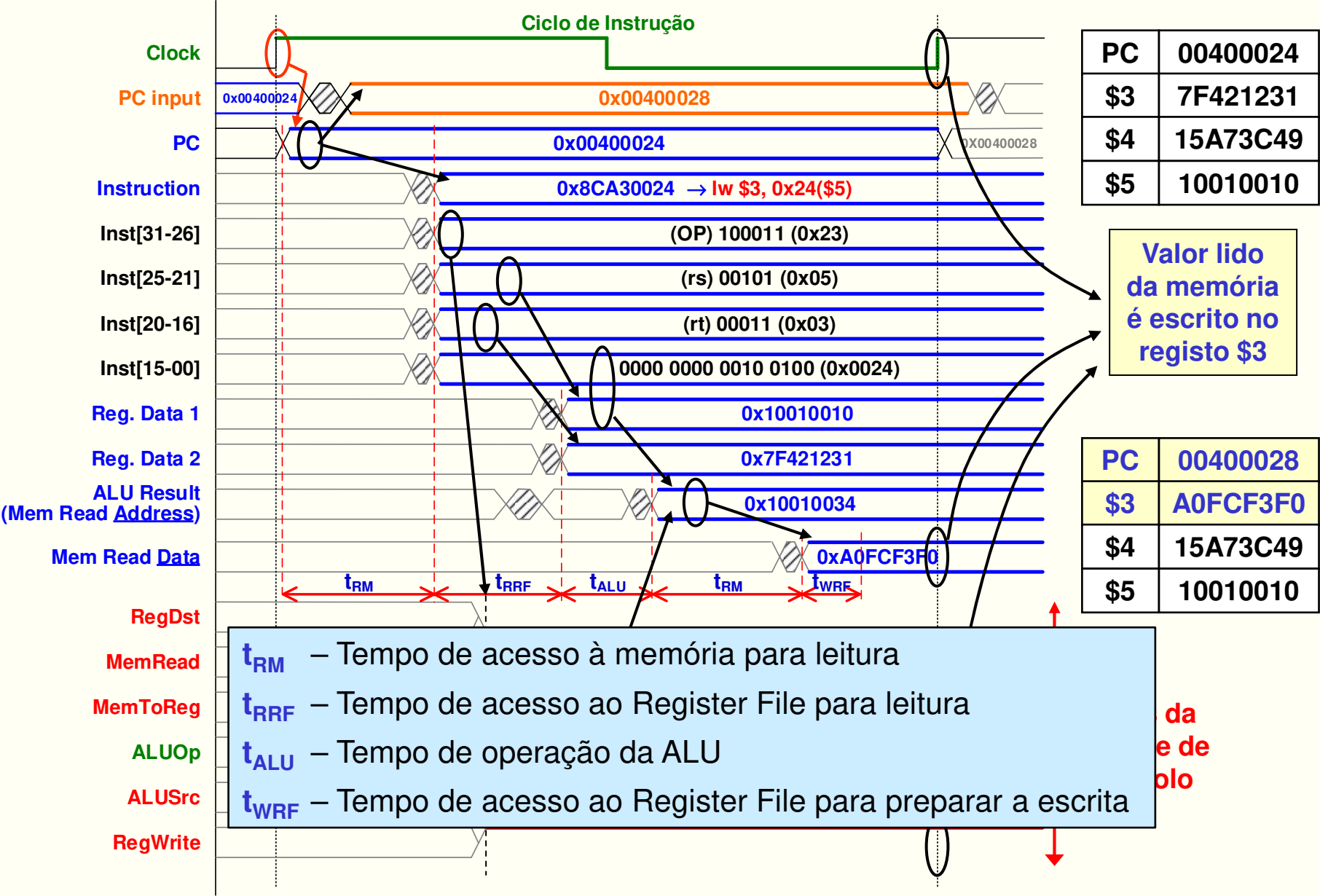
offset

100011

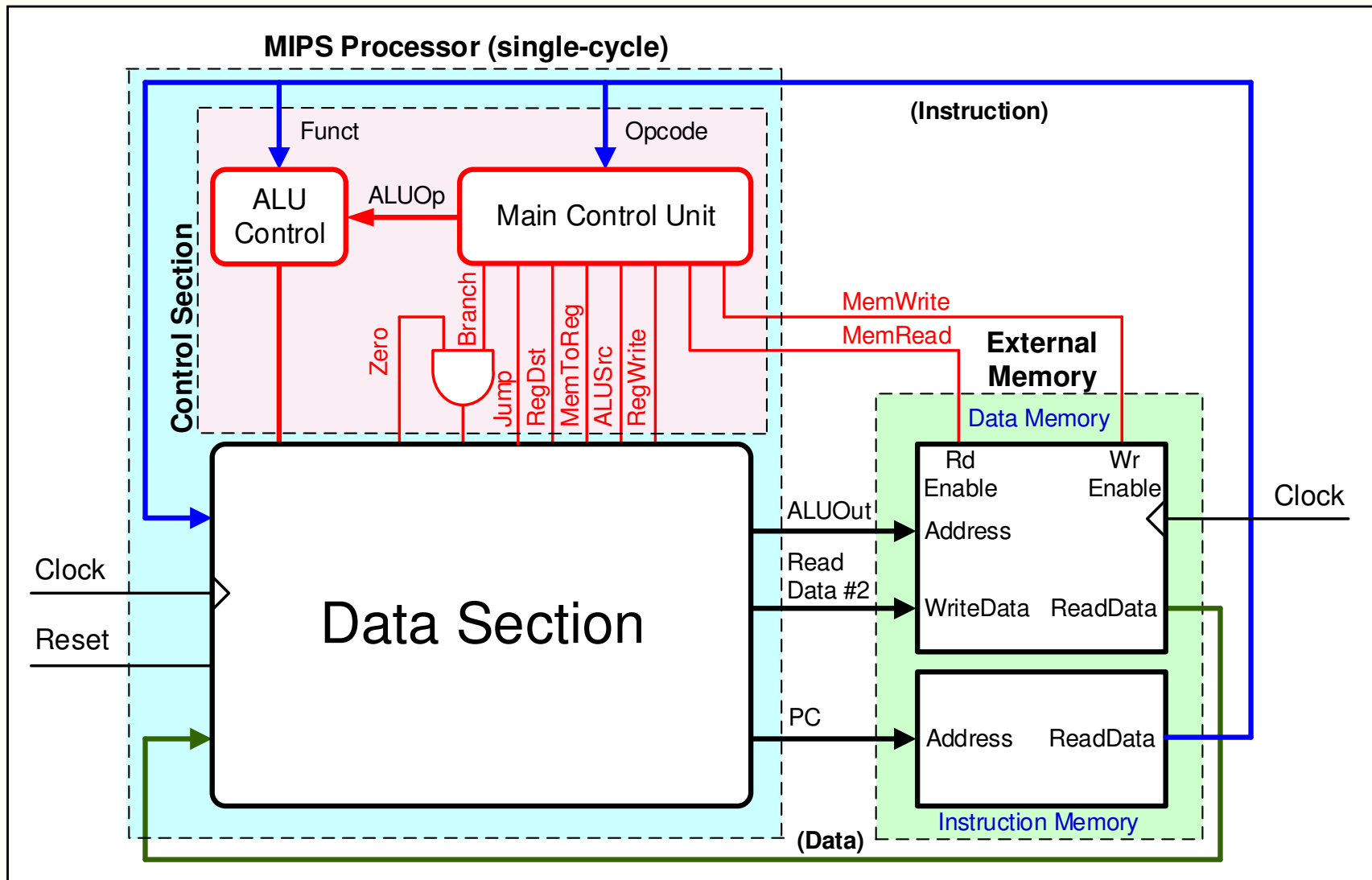
00101

00011

0000000000100100



Visão global do processador



Exercícios

- De que tipo é a unidade de controlo principal do *datapath single-cycle*?
- Como calcularia o tempo mínimo necessário para executar cada uma das instruções anteriormente analisadas?
- O que limita a frequência máxima do relógio do *datapath single-cycle*?
- Que alterações é necessário fazer ao *datapath single-cycle* para permitir a execução das instruções:
 - **"bne"** – branch not equal
 - **"jal"** – jump and link
 - **"jr"** – jump register
 - **"nor"**, **"xor"** e **"sltu"** (todas tipo R)
- Analise o *datapath* e identifique que instruções deixariam de funcionar corretamente se a unidade de controlo bloqueasse o sinal **RegWrite** a '1'.
- Repita o exercício anterior para cada uma das seguintes situações:
RegWrite='0', MemRead='0', MemWrite='0', ALUOp="00",
RegDst='1', ALUSrc='0', MemtoReg='0', MemtoReg='1'
- Que consequência teria para o funcionamento do *datapath* o bloqueio do sinal **Branch** a '1'?