Aulas 19, 20 e 21

- Limitações das arquiteturas *single-cycle*
- Versão de referência de uma arquitetura *multi-cycle*
- Exemplos de funcionamento numa arquitetura *multi-cycle*:
  - Instruções tipo R
  - Acesso à memória – LW
  - Salto condicional – BEQ
  - Salto incondicional – J
- Unidade de controlo para o *datapath multi-cycle*
  - Diagrama de estados da unidade de controlo
- Sinais de controlo e valores do *datapath multi-cycle*
  - Exemplo com execução sequencial de três instruções

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

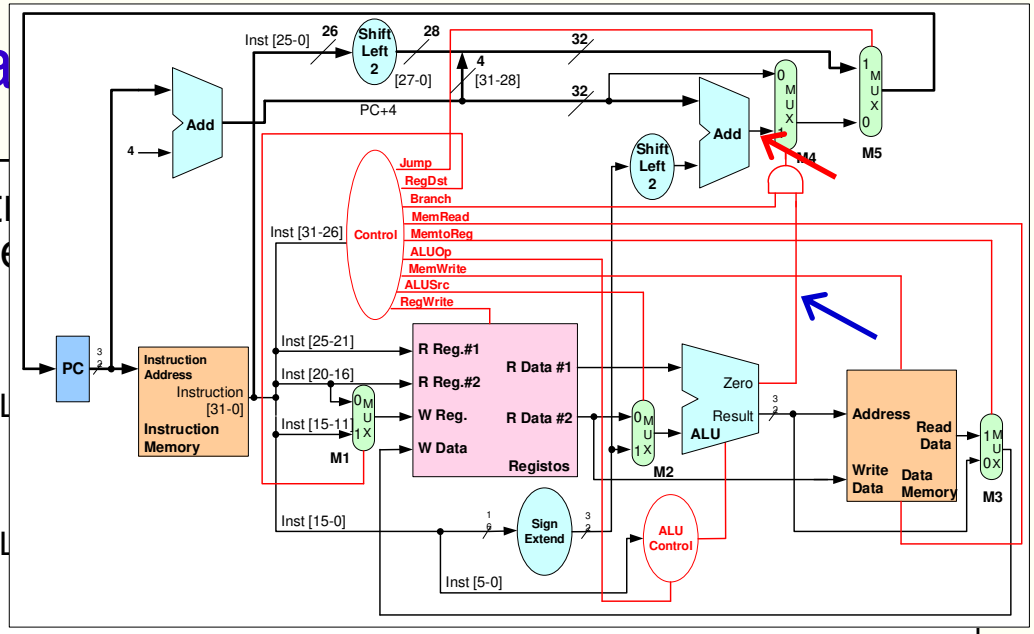
Tempo de execução das instruções (*single-cycle*)

- A **frequência máxima** do relógio de sincronização do *datapath single-cycle* está limitada pelo **tempo de execução da instrução "mais longa"**
- O **tempo de execução** de uma instrução corresponde ao **somatório dos atrasos introduzidos por cada um dos elementos operativos envolvidos na sua execução**
- Note-se que apenas os elementos operativos que se encontram em **série** contribuem para aumentar o tempo necessário para concluir a execução da instrução (caminho crítico)

Tempo de execução das instruções

- Consideremos os seguintes tempos de atraso introduzidos por cada um dos elementos operativos do *datapath single-cycle*:
  - Acesso à memória para leitura - t_{RM}
  - Acesso à memória para preparar a escrita - t_{WM}
  - Acesso ao *register file* para leitura - t_{RRF}
  - Acesso ao *register file* para preparar a escrita - t_{WRF}
  - Operação da ALU - t_{ALU}
  - Operação de um somador - t_{ADD}
  - Unidade de controlo - t_{CNTL}
  - Extensor de sinal - t_{SE}
  - Shift Left 2 - t_{SL2}
  - Tempo de *setup* do PC - t_{stPC}

Tempo de execução da



- Considerando os tempos de at
várias instruções suportadas pe

Instruções tipo R:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}$

Instrução SW:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}$

Instrução LW:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{RM} + t_{WRF}$

Instrução BEQ:

- $t_{EXEC} = t_{RM} + \max(\underbrace{\max(t_{RRF}, t_{CNTL}) + t_{ALU}}_{\text{comparação}}, \underbrace{t_{SE} + t_{SL2} + t_{ADD}}_{\text{cálculo do BTA}}) + t_{stPC}$

Instrução J:

- $t_{EXEC} = t_{RM} + \max(t_{CNTL}, t_{SL2}) + t_{stPC}$

Notas:

1. Considera-se que o tempo de cálculo de PC+4 é muito inferior ao somatório dos restantes tempos envolvidos na execução da instrução
2. O tempo t_{CNTL} inclui o tempo de atraso da unidade de controlo da ALU
3. Desprezam-se os tempos de atraso introduzidos pelos *multiplexers*
4. Só se considera o t_{stPC} nas instruções de controlo de fluxo.

Tempo de execução das instruções

- Considerando os tempos de atraso anteriores, os tempos de execução das várias instruções suportadas pelo datapath single cycle serão:

Instruções tipo R:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}) + t_{ALU} + t_{WRF}$

Instrução SW:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{WM}$

Instrução LW:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{RM} + t_{WRF}$

Instrução BEQ:

- $t_{EXEC} = t_{RM} + \max(\mathbf{\max(t_{RRF} , t_{CNTL}) + t_{ALU}}, t_{SE} + t_{SL2} + t_{ADD}) + t_{stPC}$

comparação

cálculo do BTA

Instrução J:

- $t_{EXEC} = t_{RM} + \max(t_{CNTL}, t_{SL2}) + t_{stPC}$

Notas:

1. Considera-se que o tempo de cálculo de PC+4 é muito inferior ao somatório dos restantes tempos envolvidos na execução da instrução
2. O tempo t_{CNTL} inclui o tempo de atraso da unidade de controlo da ALU
3. Desprezam-se os tempos de atraso introduzidos pelos *multiplexers*
4. Só se considera o t_{stPC} nas instruções de controlo de fluxo.

Tempo de execução das instruções - exemplo

- Considerem-se os seguintes valores hipotéticos para os tempos de atraso introduzidos por cada um dos elementos operativos do *datapath single-cycle*:
  - Acesso à memória para leitura (t_{RM}): 5ns
  - Acesso à memória para preparar escrita (t_{WM}): 5ns
  - Acesso ao *register file* para leitura (t_{RRF}): 3ns
  - Acesso ao *register file* para preparar escrita (t_{WRF}): 3ns
  - Operação da ALU (t_{ALU}): 4ns
  - Operação de um somador (t_{ADD}): 1ns
  - *Multiplexers* e restantes elementos operativos: 0ns
  - Unidade de controlo (t_{CNTL}): 1ns
  - Tempo de *setup* do PC (t_{stPC}): 1ns

Tempo de execução das instruções - exemplo

- **Instruções tipo R:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}) + t_{ALU} + t_{WFR}$
    $= 5 + \max(3, 1) + 4 + 3 =$ **15 ns**
- **Instrução SW:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{WM}$
    $= 5 + \max(3, 1, 0) + 4 + 5 =$ **17 ns**
- **Instrução LW:**
  - $t_{EXEC} = t_{RM} + \max(t_{RFR}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{RM} + t_{WFR}$
    $= 5 + \max(3, 1, 0) + 4 + 5 + 3 =$ **20 ns**
- **Instrução BEQ:**
  - $t_{EXEC} = t_{RM} + \max(\max(t_{RFR} , t_{CNTL}) + t_{ALU}, t_{SE} + t_{SL2} + t_{ADD}) + t_{stPC}$
    $= 5 + \max(\max(3, 1) + 4, 0 + 0 + 1) + 1 =$ **13 ns**
- **Instrução J:**
  - $t_{EXEC} = t_{RM} + \max(t_{CNTL}, t_{SL2}) + t_{stPC} = 5 + \max(1, 0) + 1 =$ **7 ns**

Limitações das soluções *single-cycle*

- Face à análise anterior, a máxima frequência de trabalho seria:

  fmax = 1 / 20ns = 50MHz

- Com a mesma tecnologia, contudo, uma multiplicação ou divisão poderia demorar um tempo da ordem dos 150ns
- Para poder suportar uma ALU com capacidade para efetuar operações de multiplicação/divisão, a frequência de relógio máxima do nosso *datapath* baixaria para 6.66MHz
- Esta frequência máxima limitaria drasticamente a eficiência do *datapath*, mesmo que as instruções de multiplicação ou divisão sejam raramente utilizadas
- Uma solução ingénua, seria usar um relógio de frequência variável, ajustável em função da instrução que está em execução – é uma solução tecnicamente inviável

Limitações das soluções *single-cycle*

- O tempo de execução de um programa pode ser calculado como:

$$Texec_{CPU} = \#\,Instruções \times CPI \times Clock\_Cycle_{CPU}$$

  sendo **CPI o número médio de ciclos de relógio por instrução** na execução do programa em causa; no caso da implementação *single-cycle* o CPI é 1, logo:

$$Texec_{CPU} = \#\,Instruções \times Clock\_Cycle_{CPU}$$

- Define-se ainda:

$$Desempenho_{CPU} = 1/TexecCPU$$

- O desempenho de um CPU ($CPU_{ANALISE}$) relativamente a outro ($CPU_{REFERENCIA}$) pode ser expresso por:

$$\frac{Desempenho_{CPU\_ANALISE}}{Desempenho_{CPU\_REFERENCIA}} = \frac{Texec_{CPU\_REFERENCIA}}{Texec_{CPU\_ANALISE}}$$

Limitações das soluções *single-cycle*

- **Exercício**: calcular o ganho de desempenho que se obteria com uma implementação de *clock* variável relativamente a uma com o *clock* fixo, na execução de um programa com o seguinte *mix* de instruções:
  - 20% de lw, 10% de sw, 50% de tipo R, 15% de branches e 5% de jumps
  - assumindo os tempos execução determinados anteriormente para os vários tipos de instruções (LW: 20ns, SW: 17ns, R-Type: 15ns, BEQ: 13ns, J: 7ns)
- Para este exemplo, o tempo médio de execução de cada instrução num CPU com *clock* variável seria calculado como:

$$T_{MED\_INSTR} = 0{,}2*20 + 0{,}1*17 + 0{,}5*15 + 0{,}15*13 + 0{,}05*7 = 15{,}5ns$$

- O ganho de desempenho do CPU com *clock* variável relativamente a um com *clock* fixo seria então:

$$\frac{Des_{CPU\_CLOCK\_VARIAVEL}}{Des_{CPU\_CLOCK\_FIXO}} = \frac{\#\,Instruções \times 20}{\#\,Instruções \times T_{MED\_INSTR}} = 1{,}29$$

A implementação com *clock* variável não é viável mas permite entender o que está a ser sacrificado quando todas as instruções têm que ser executadas num único ciclo de relógio com tempo fixo

Limitações das soluções *single-cycle* - conclusões

- Num *datapath* que suporte instruções com complexidade variável, é a **instrução mais lenta que determina a máxima frequência de trabalho**, mesmo que seja uma instrução pouco frequente
- Uma vez que o ciclo de relógio é igual ao maior tempo de atraso de todas as instruções, não é útil usar técnicas que reduzam o atraso do caso mais comum mas que não melhorem o maior tempo de atraso
  - Isto contraria um dos princípios-chave de desenho: *make the common case fast* (o que é mais comum deve ser mais rápido)
- Elementos operativos que estejam envolvidos na execução de uma instrução **não podem ser usados para mais do que uma operação por ciclo de relógio** (ex: memória de instruções e de dados, ALU e somadores, ...)

O *datapath Multi-cycle*

- Uma solução para os problemas enumerados passa por abdicar do princípio de que todas as instruções devem ser executadas num único ciclo de relógio
- Em alternativa, as várias instruções que compõem o *set* de instruções podem ser executadas em vários ciclos de relógio (*multi-cycle)*:
  - A execução da instrução é decomposta num conjunto de operações
  - Em cada ciclo de relógio poderão ser realizadas **várias operações, desde que sejam independentes** (por exemplo, *instruction fetch* e cálculo de PC+4 ou *operand fetch* e cálculo do BTA)
  - Cada uma dessas operações faz uso de um elemento operativo fundamental: memória, *register file* ou ALU
- Desta forma, o período de relógio fica apenas limitado pelo maior dos tempos de atraso de cada um dos elementos operativos fundamentais
- Para os tempos de atraso que considerámos anteriormente, a máxima frequência de relógio seria assim: **fmax = 1 / t_{RM} = 1 / 5ns = 200MHz**

O *datapath Multi-cycle*

- A arquitetura *multi-cycle* do MIPS que vamos analisar adota um ciclo de instrução composto por um **máximo de cinco passos distintos**, cada um deles executado em 1 ciclo de relógio
- A distribuição das operações por estes 5 passos tenta distribuir equitativamente o trabalho a realizar em cada ciclo
- Na definição destes passos pressupõe-se que durante um ciclo de relógio apenas é possível efetuar uma das seguintes ações fundamentais da execução de uma instrução:
  - Acesso à memória externa (uma leitura ou uma escrita)
  - Acesso ao *Register File* (uma leitura ou uma escrita)
  - Operação na ALU
- No **mesmo ciclo de relógio**, podem ser realizadas operações em elementos operativos distintos, desde que sejam independentes
  - Exemplos: **um acesso à memória externa e uma operação na ALU**, ou um **acesso ao *Register File* e uma operação na ALU**

Alternativa às soluções *single-cycle*

- Uma outra vantagem duma solução de execução em vários ciclos de relógio (*multi-cycle*) é que um **mesmo elemento operativo** pode ser utilizado mais do que uma vez, no contexto da execução de uma mesma instrução, desde que em ciclos de relógio distintos:
  - A memória externa poderá ser partilhada por instruções e dados
  - A mesma ALU poderá ser usada, para além das operações que já realizava na implementação *single-cycle*, para:
    - Calcular o valor de PC+4
    - Calcular o endereço alvo das instruções de salto condicional (BTA)
- A versão ***multi-cycle*** passará assim a ter:
  - **Uma única memória** para programa e dados (arquitetura Von Neumann)
  - **Uma única ALU**, em vez de uma ALU e dois somadores

O *datapath Multi-cycle* – fases de execução

Fase 1 (memória, ALU):

- *Instruction fetch* **e** cálculo de PC+4

Fase 2 (*register file*, ALU, unidade de controlo):

- *Operand fetch* **e** cálculo do *branch target address* **e** *Instruction decode*

Fase 3 (ALU):

- Execução da operação na ALU (instruções tipo R / addi / slti), **ou**
- Cálculo do endereço de memória (instr. de acesso à memória), **ou**
- Comparação dos operandos - instrução *branch* (conclusão da instrução)

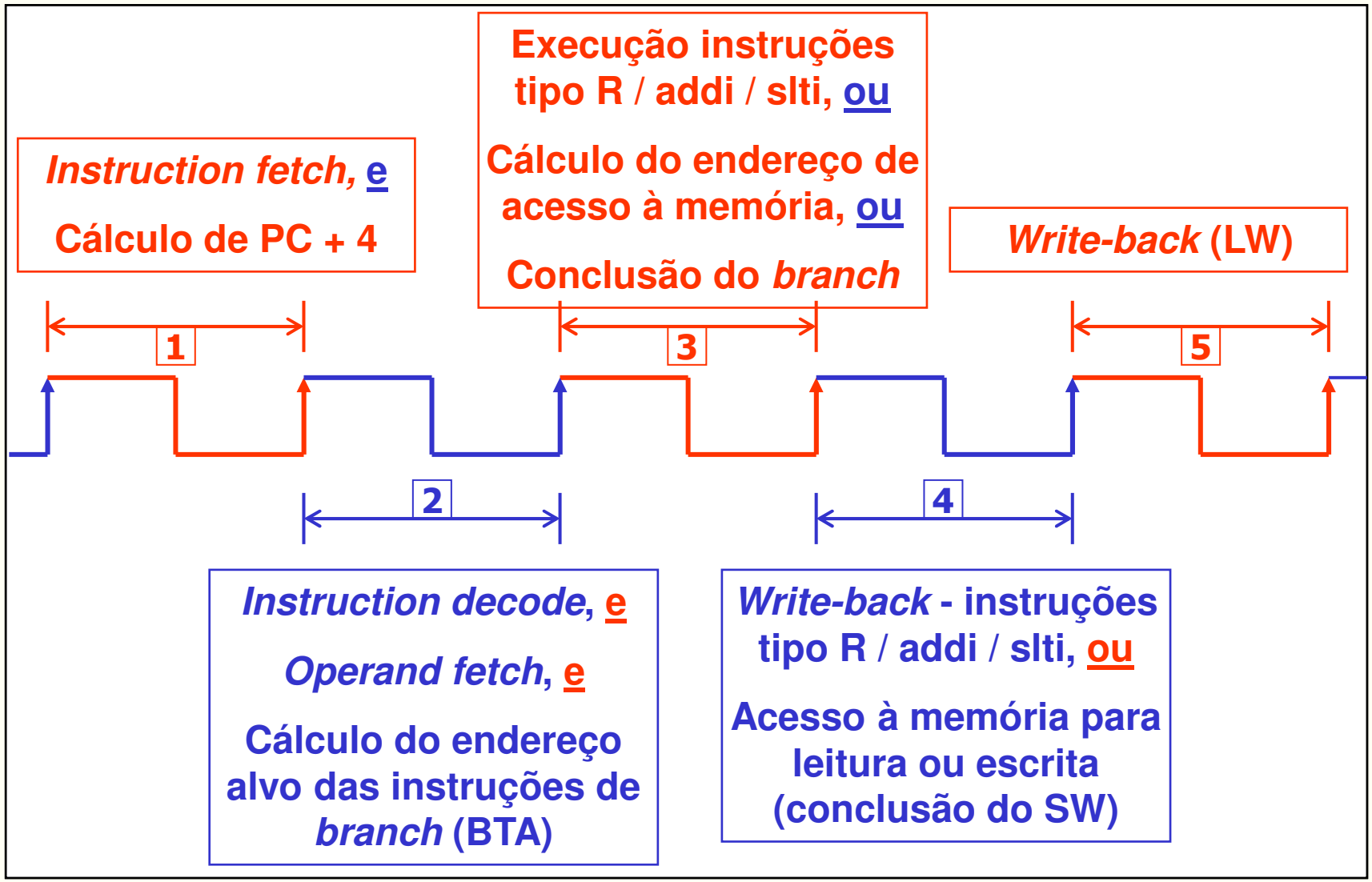
Fase 4 (memória, *register file*):

- Acesso à memória para leitura (instrução LW), **ou**
- Acesso à memória para escrita (conclusão da instrução SW), **ou**
- Escrita no *Register File* (conclusão das instruções tipo R / addi / slti: ***write-back***)

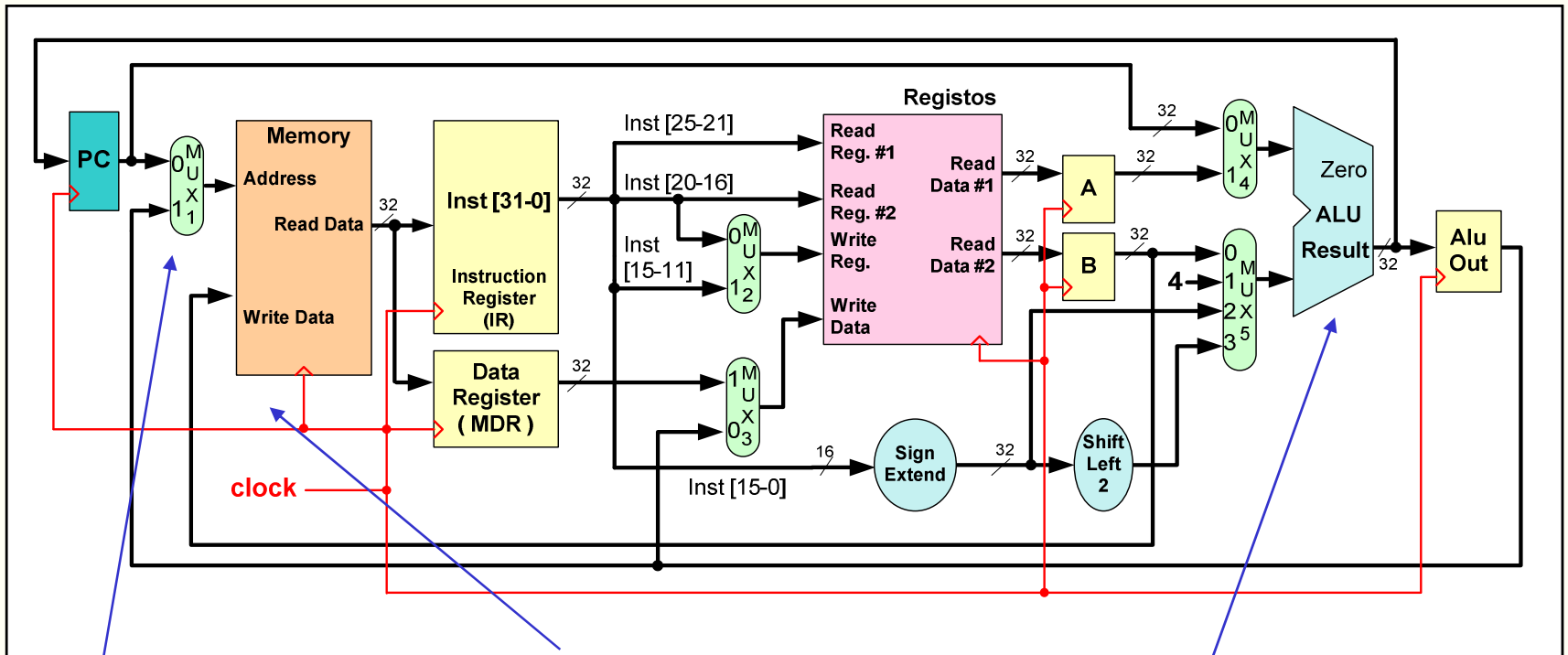
Fase 5 *(register file):*

- Escrita no *Register File* (conclusão da instrução LW: ***write-back***)

O *datapath Multi-cycle* – fases de execução

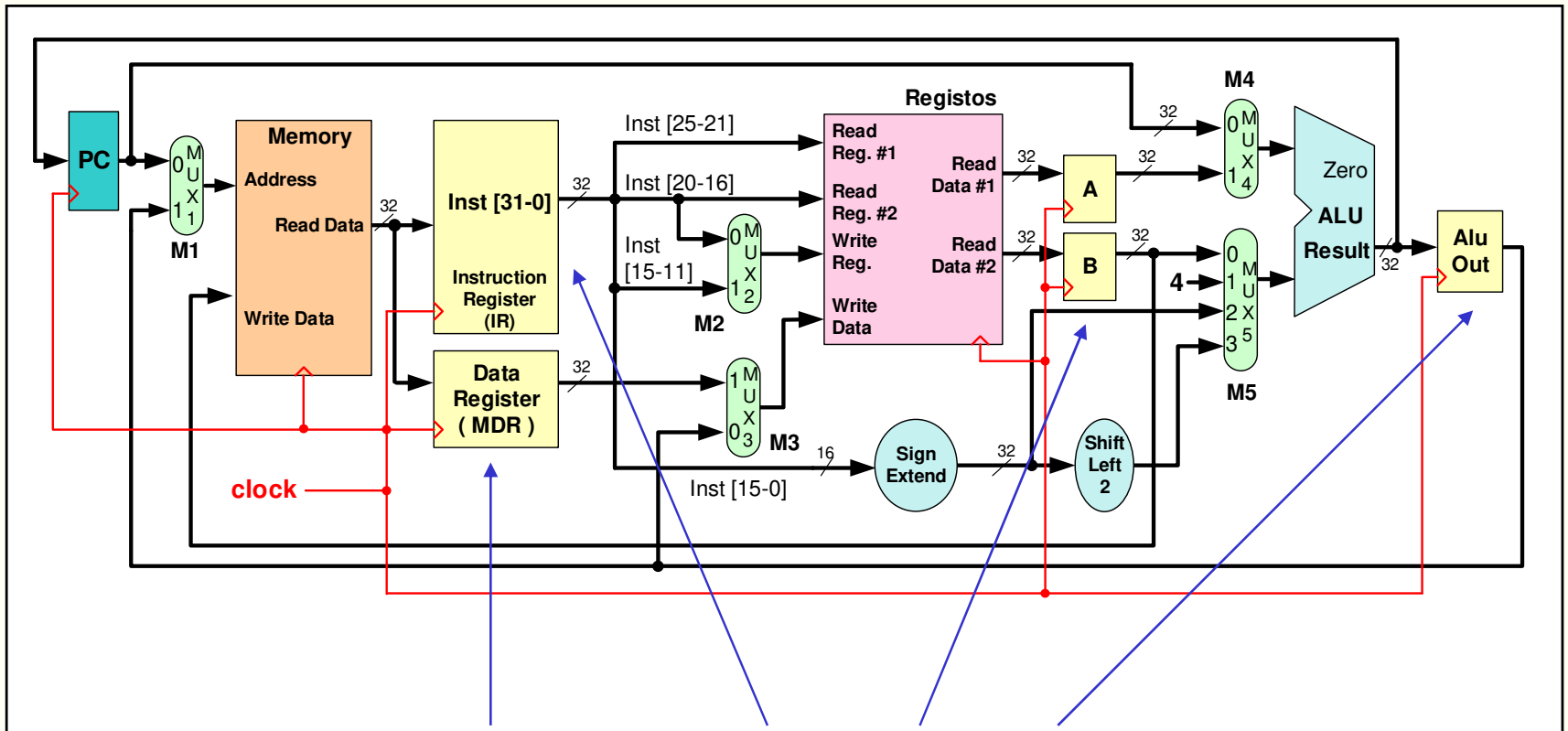


O *datapath Multi-cycle* (sem BEQ e J)



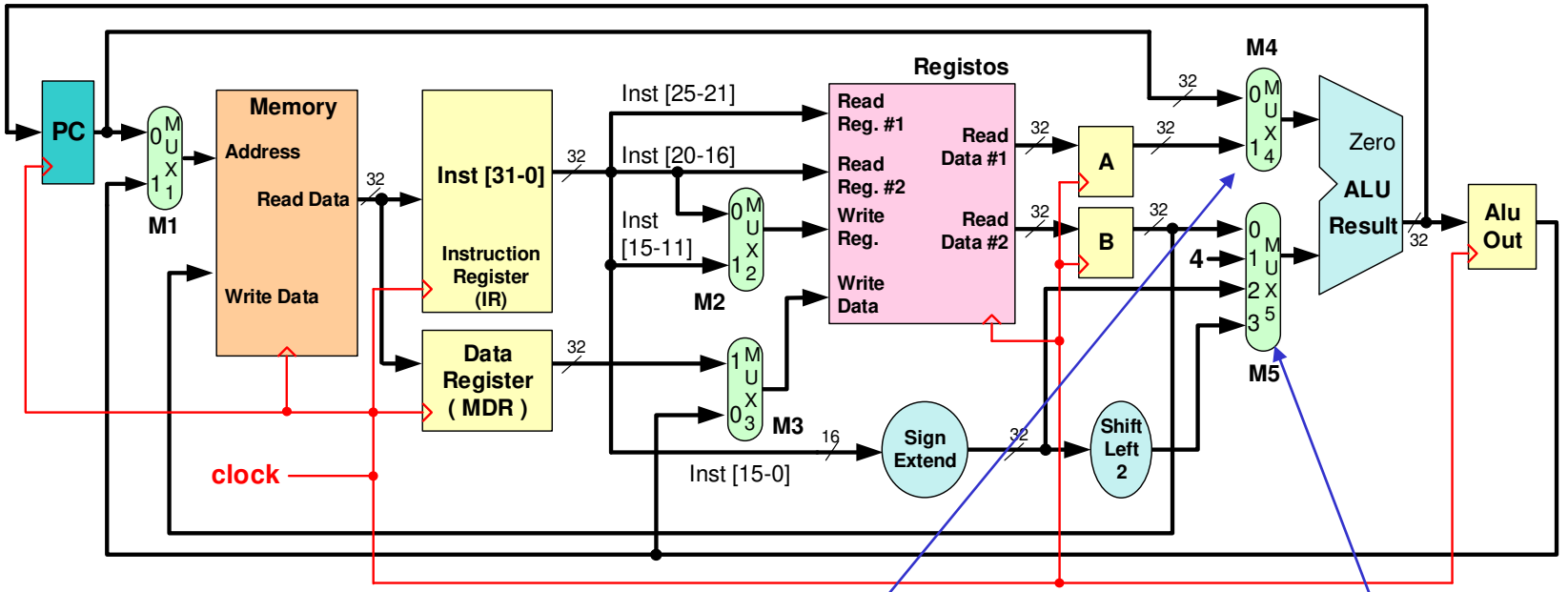
- **Uma única memória para programa e dados**
  - Um *multiplexer* no barramento de endereços da memória permite selecionar o endereço a usar:
    - o conteúdo do PC (para leitura da instrução) ou
    - o valor calculado na ALU (para acesso de leitura/escrita de dados nas instruções LW/SW)
- **Uma única ALU** (em vez de uma ALU e dois somadores)

O *datapath Multi-cycle* (sem BEQ e J)



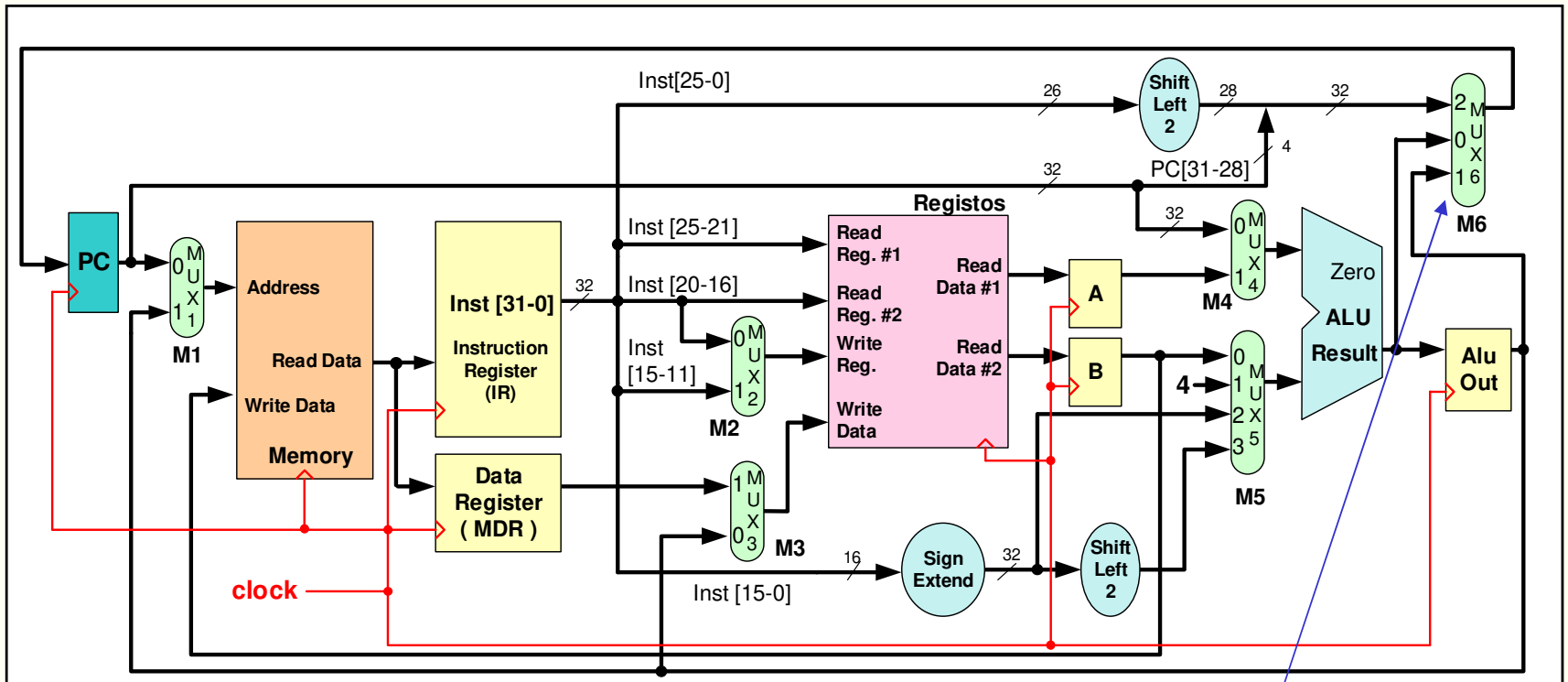
- Registos adicionados à saída dos elementos operativos fundamentais para armazenamento da informação obtida/calculada durante o ciclo de relógio corrente e que será utilizada no ciclo de relógio seguinte

O *datapath Multi-cycle* (sem BEQ e J)



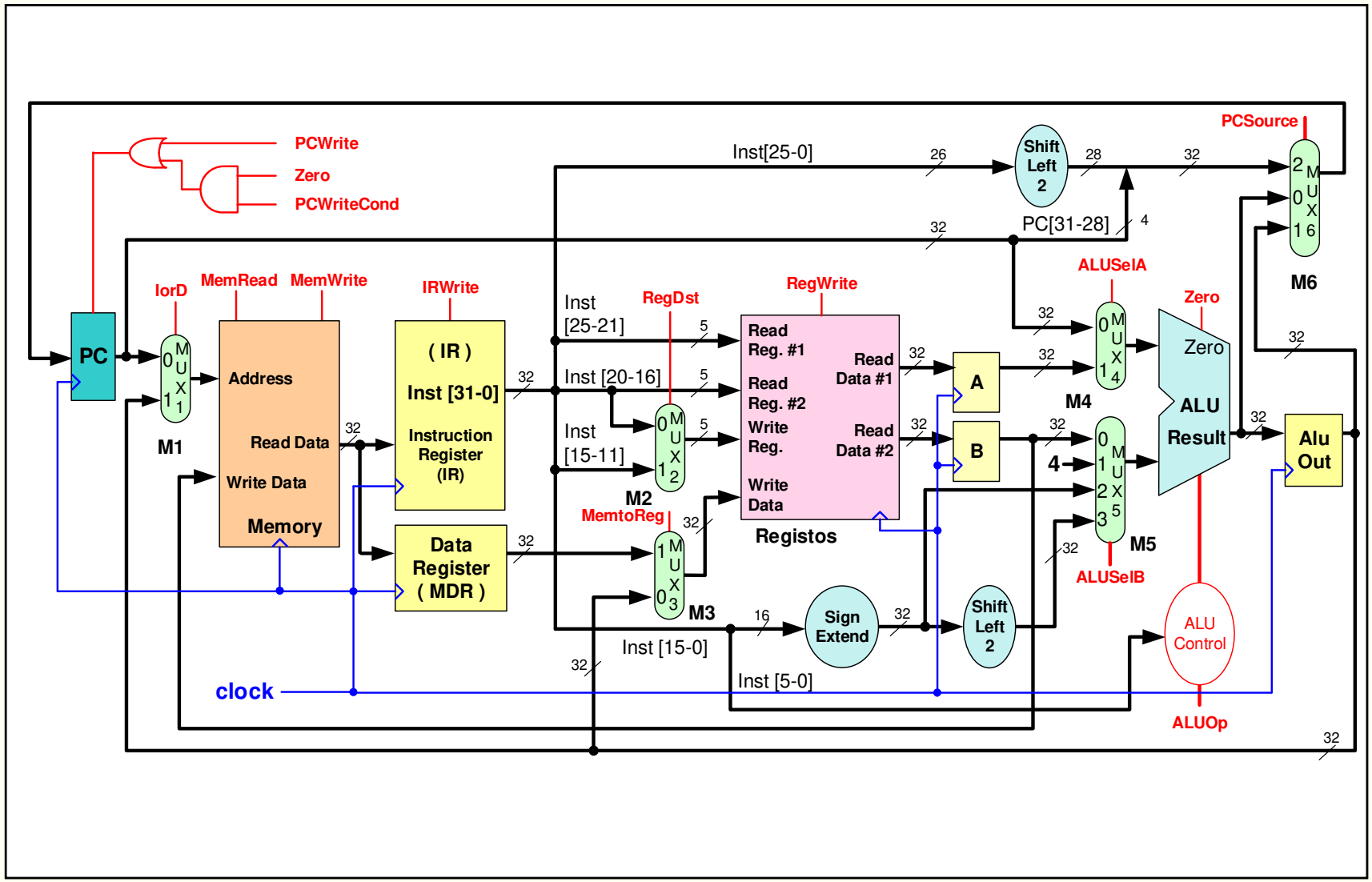
- A utilização de uma única ALU obriga às seguintes alterações nas suas entradas:
  - Um *multiplexer* adicional na primeira entrada, que encaminha a saída do registo A ou a saída do registo PC
  - O *multiplexer* da segunda entrada é aumentado para poder suportar o incremento do PC (constante 4) e o cálculo do endereço alvo das instruções de branch (BTA - *branch target address*)

O *datapath Multi-cycle* com as instruções BEQ e J



- Com as instruções de salto, o **registo PC** pode ser atualizado com um dos valores:
  - A **saída da ALU** que contém o PC+4 calculado durante o *instruction fetch* (na 1ª fase)
  - A saída do registo **ALUOut** que armazena o endereço alvo das instruções de *branch* (BTA) calculado na ALU (na 2ª fase)
  - ***Jump Target Address*** - 26 LSB da instrução multiplicados por 4 (*shift left* 2) concatenados com os 4 MSB do PC atual (o PC foi já incrementado na 1ª fase)

O *datapath Multi-cycle*, com os sinais de controlo



O *datapath Multi-cycle* – sinais de controlo

Sinal	Efeito quando não activo ('0')	Efeito quando activo ('1')
MemRead	Nenhum (barramento de dados da memória em alta impedância)	O conteúdo da memória no endereço indicado é apresentado à saída
MemWrite	Nenhum	O conteúdo do registo de memória, 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
IRWrite	Nenhum	O valor lido da memória externa é escrito no Instruction Register
PCWrite	Nenhum	O PC é atualizado **incondicionalmente** na próxima transição ativa do sinal de relógio
PCWriteCond	Nenhum	O PC é atualizado **condicionalmente** na próxima transição ativa do relógio
ALUSelA	O primeiro operando da ALU é o PC	O primeiro operando da ALU provém do registo indicado no campo rs
RegDst	O endereço do registo destino provém do campo rt	O endereço do registo destino provém do campo rd
MemtoReg	O valor apresentado para escrita no registo destino provém da ALU	O valor apresentado na entrada de dados do Register File provém do Data Register
IorD	O PC é usado para fornecer o endereço à memória externa	A saída do registo AluOut é usada para providenciar um endereço para a memória externa

O *datapath Multi-cycle* – sinais de controlo

Sinal	Valor	Efeito
ALUSelB	00	A segunda entrada da ALU provém do registo indicado pelo campo rt
	01	A segunda entrada da ALU é a constante 4
	10	A segunda entrada da ALU é a versão de sinal estendido dos 16 bits menos significativos do IR (instruction register)
	11	A segunda entrada da ALU é a versão de sinal estendido e deslocada de dois bits, dos 16 bits menos significativos do IR (instruction register)
ALUOp	00	ALU efetua uma adição
	01	ALU efetua uma subtração
	10	O campo "funct" da instrução determina qual a operação da ALU
	11	ALU efetua um SLT
PCSource	00	O valor do PC é atualizado com o resultado da ALU (IF)
	01	O valor do PC é atualizado com o resultado da AluOut (Branch)
	10	O valor do PC é atualizado com o valor target do Jump
	11	Não usado

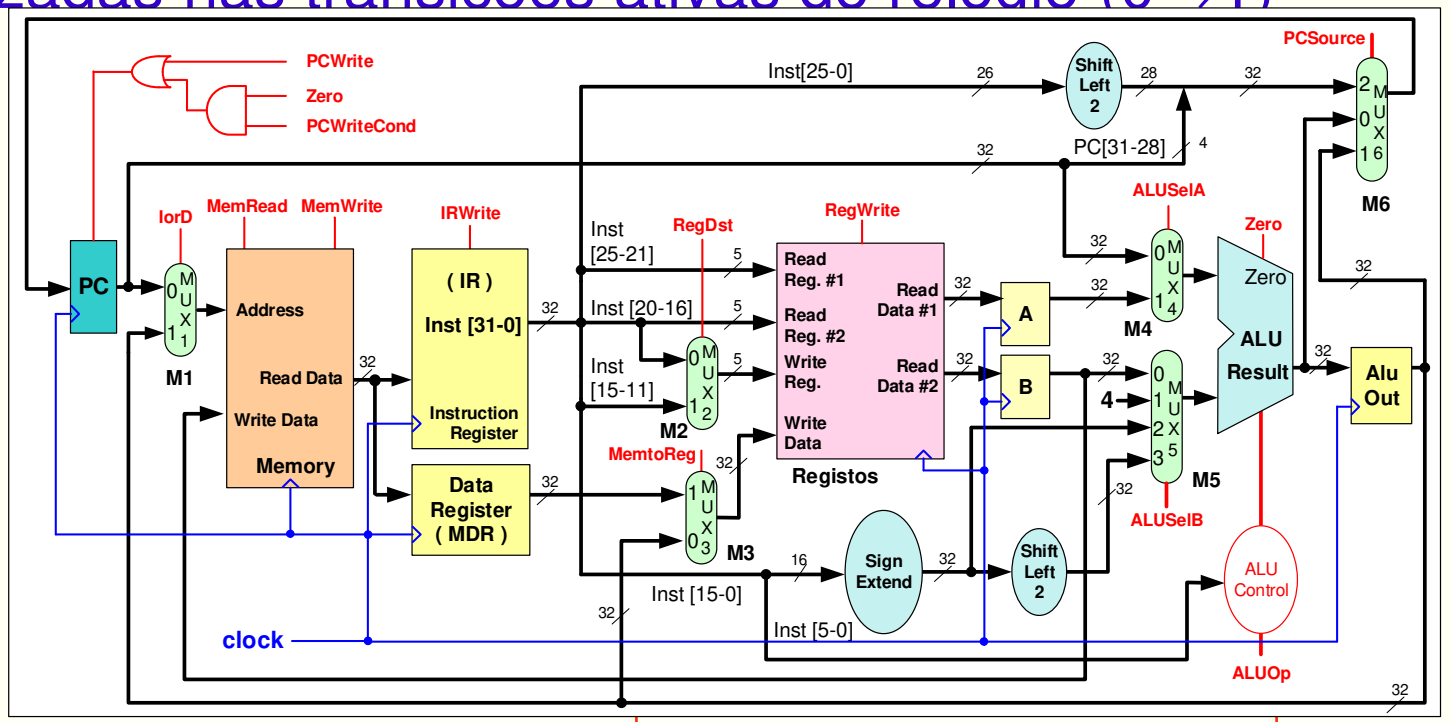
Ações realizadas nas transições ativas do relógio (0→1)

Início da execução da instrução

IR = Memory[PC]

PC = PC + 4

1

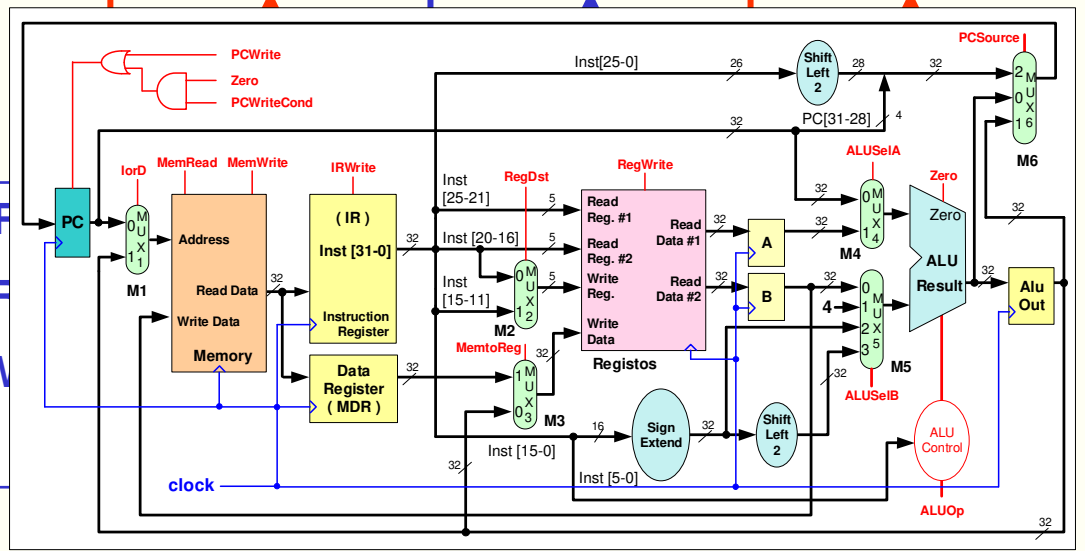


2

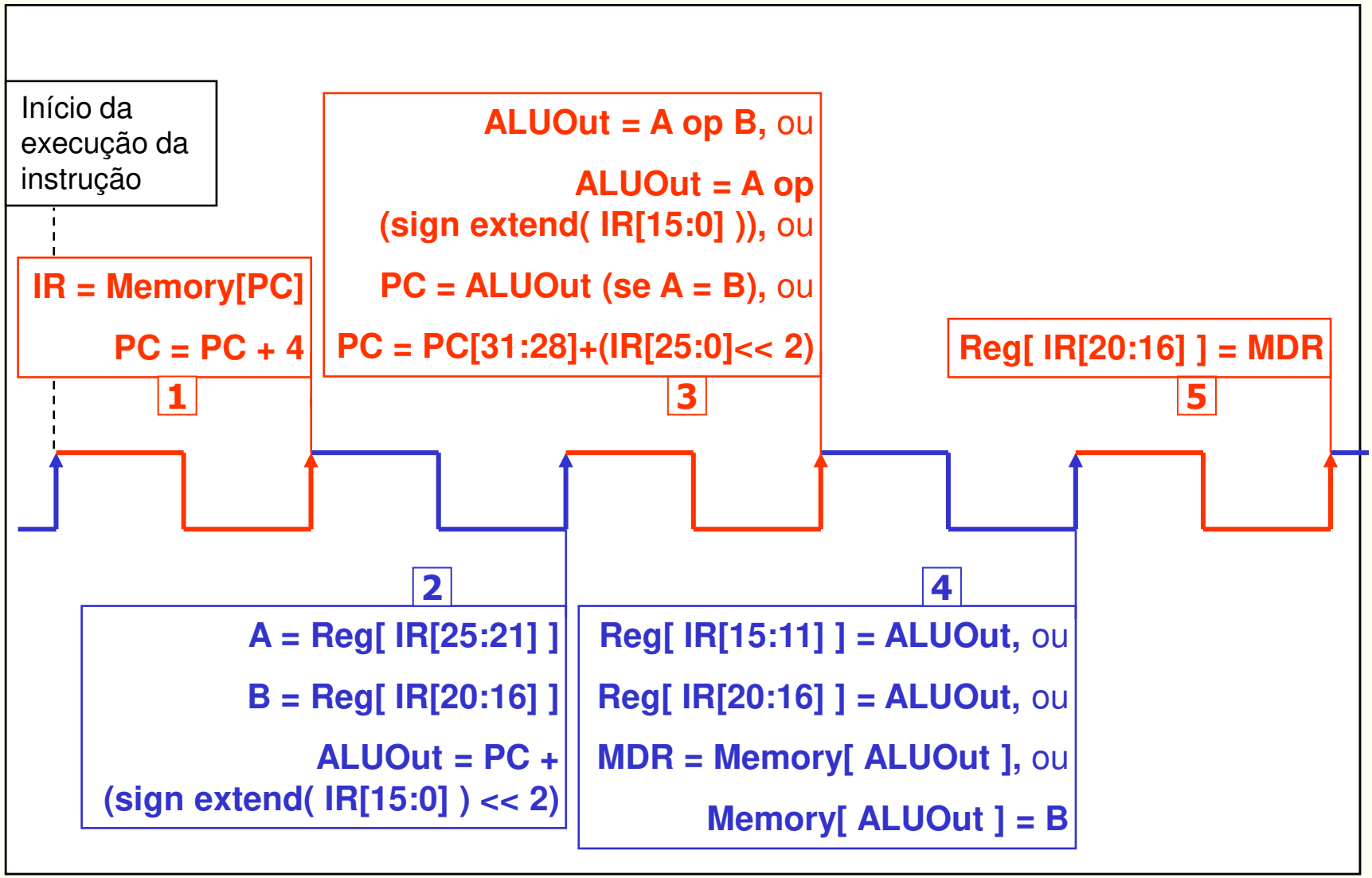
A = Reg[IR[25:21]]

B = Reg[IR[20:16]]

ALUOut = PC + (sign extend(IR[15:0]) << 2)



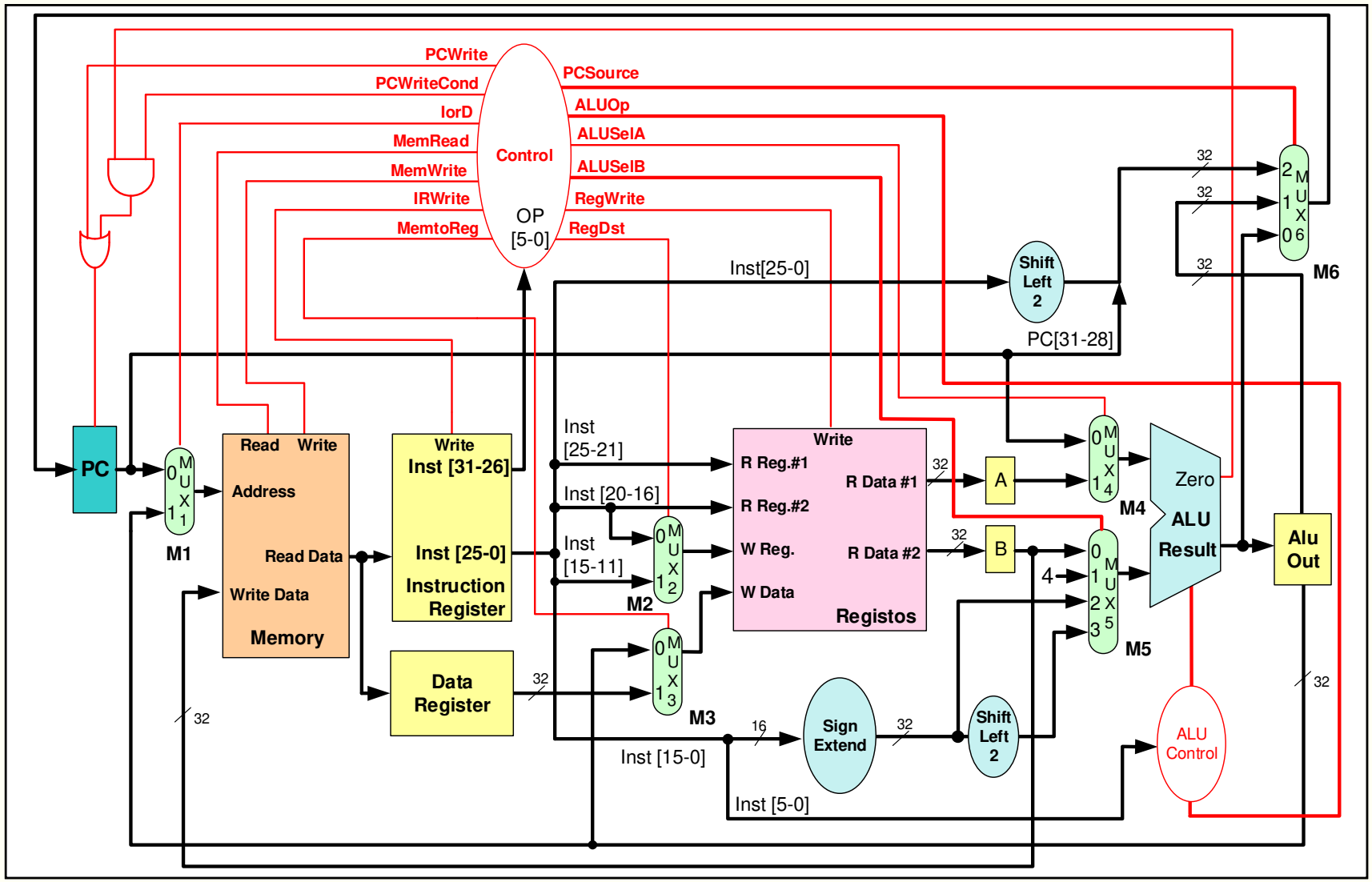
Ações realizadas nas transições ativas do relógio (0→1)



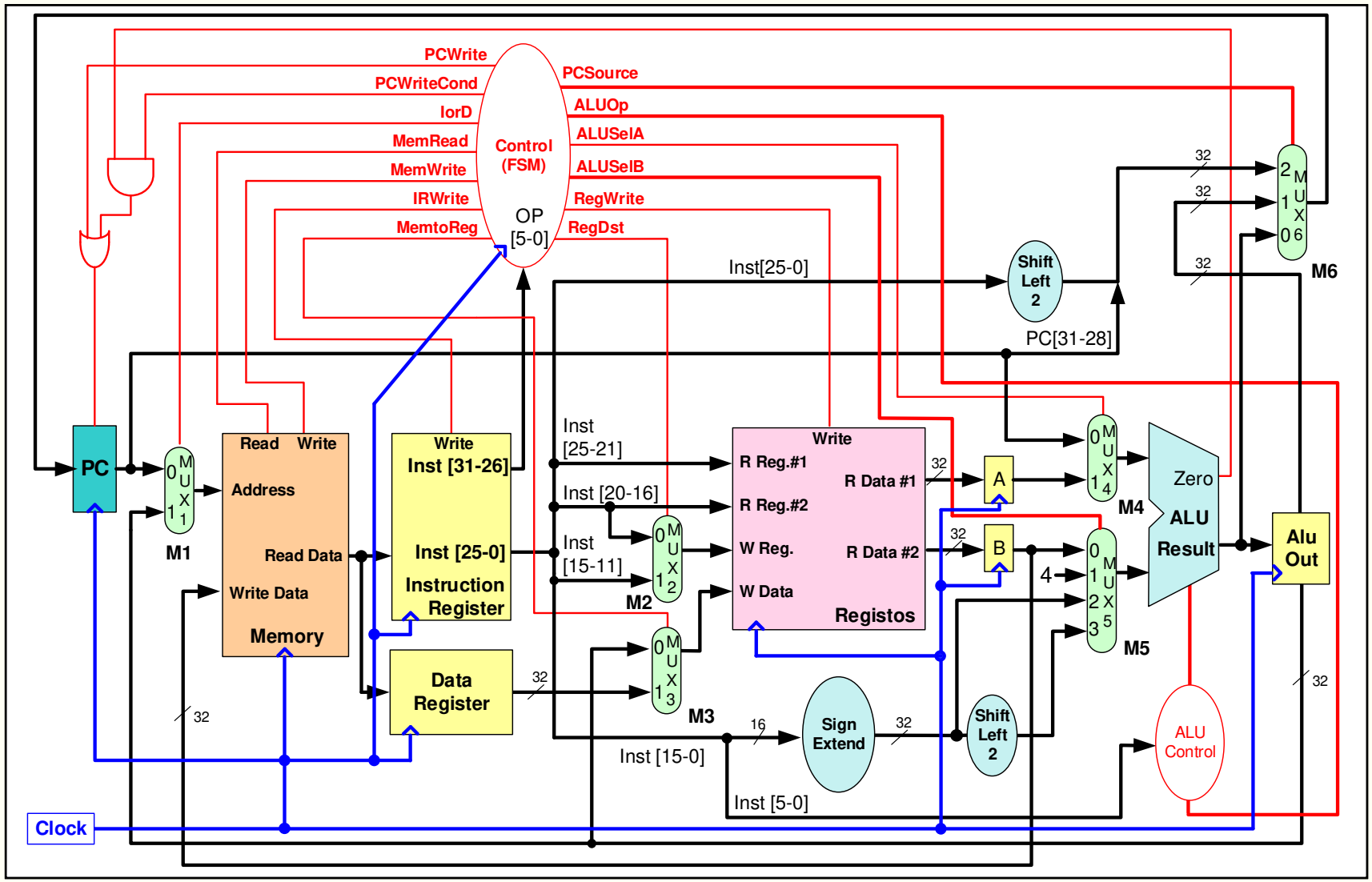
Ações realizadas nas transições ativas do relógio (0→1)

Passo	Ação p/ as R-Type / ADDI / SLTI	Ação p/ instruções que referenciam a memória	Ação p/ os branches
Instruction fetch	IR = Memory[PC] PC = PC + 4		
Instruction decode, register fetch, cálculo do BTA	A = Reg[IR[25:21]] B = Reg[IR[20:16]] ALUOut = PC + (sign extended(IR[15:0]) << 2)		
Execução (tipoR/addi/slti), cálculo de endereços ou conclusão dos branches	ALUOut = A op B ou ALUOut = A op extend(IR[15:0])	ALUOut = A+sign-extended(IR[15:0]))	If (A == B) then PC = ALUOut
Acesso à memória (leitura-LW; ou escrita-SW) ou escrita no File Register (write-back, instruções tipo R/addi/slti)	Tipo R: Reg[IR[15:11]]= ALUOut ADDI / SLTI: Reg[IR[20:16]]= ALUOut	MDR = Memory[ALUOut] ou Memory[ALUOut] = B	
Escrita no File Register (write-back, instrução LW)		Reg[IR[20:16]] = MDR	

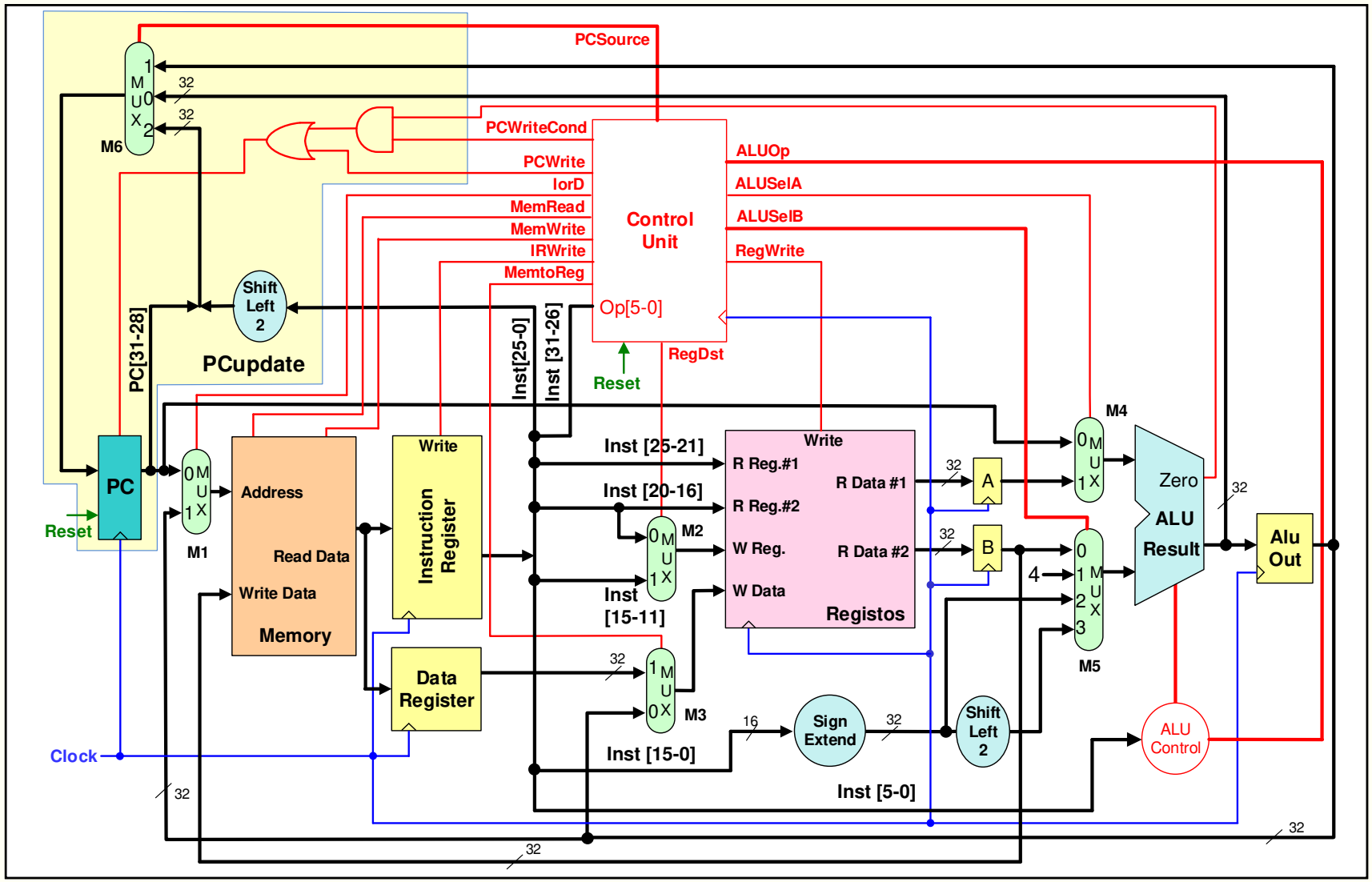
O *datapath Multi-cycle* completo



O *datapath Multi-cycle* completo



Módulo de atualização do PC



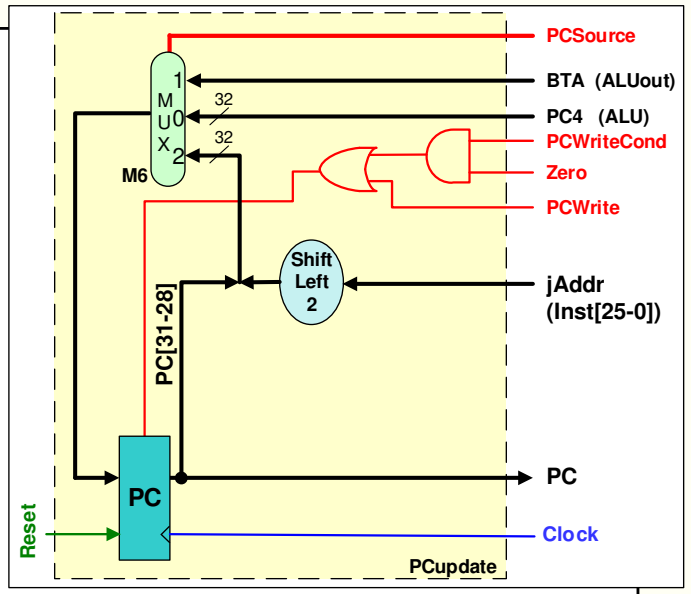
Módulo de atualização do PC – VHDL

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

entity PCupdate is
 port(clk      : in  std_logic;
      reset    : in  std_logic;
      zero     : in  std_logic;
      PCSource: in  std_logic_vector(1 downto 0);
      PCWrite  : in  std_logic;
      PCWriteCond : in std_logic;
      PC4      : in  std_logic_vector(31 downto 0);
      BTA      : 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 – VHDL

```
architecture Behavioral of PCupdate is
  signal s_pc : std_logic_vector(31 downto 0);
  signal s_pcEnable : std_logic;
begin
  s_pcEnable <= PCWrite or (PCWriteCond and zero);
  process(clk)
  begin
      if(rising_edge(clk)) then
          if(reset = '1') then
              s_pc <= (others => '0');
          elsif(s_pcEnable = '1') then
              case PCSource is
                  when "01" => -- BTA
                      s_pc <= BTA;
                  when "10" => -- JTA
                      s_pc <= s_pc(31 downto 28) & jAddr & "00";
                  when others => -- PC + 4
                      s_pc <= PC4;
              end case;
          end if;
      end if;
  end process;
  pc <= s_pc;
end Behavioral;
```



Exemplos de funcionamento

- Nos exemplos seguintes as cores indicam o estado, o valor ou a utilização dos sinais de controlo, barramentos e elementos de estado/combinatórios. O significado atribuído a cada cor é:
- Sinais de controlo:
  - **vermelho → 0**
  - **verde → diferente de zero**
  - **cinzento → "don't care"**
- Barramentos:
  - **azul → Relevantes no contexto do ciclo da instrução**
  - **preto → Não relevantes no contexto do ciclo da instrução**
- Elementos de estado / combinatórios:
  - fundo de cor → Usados no contexto do ciclo da instrução
  - fundo branco → Não usados no contexto do ciclo da instrução
- Elementos de estado:
  - fundo de cor com textura → Escritos no final do ciclo de relógio corrente

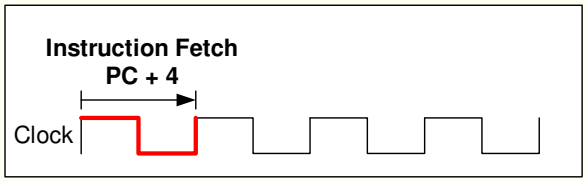
Funcionamento do *datapath* nas instruções do tipo R

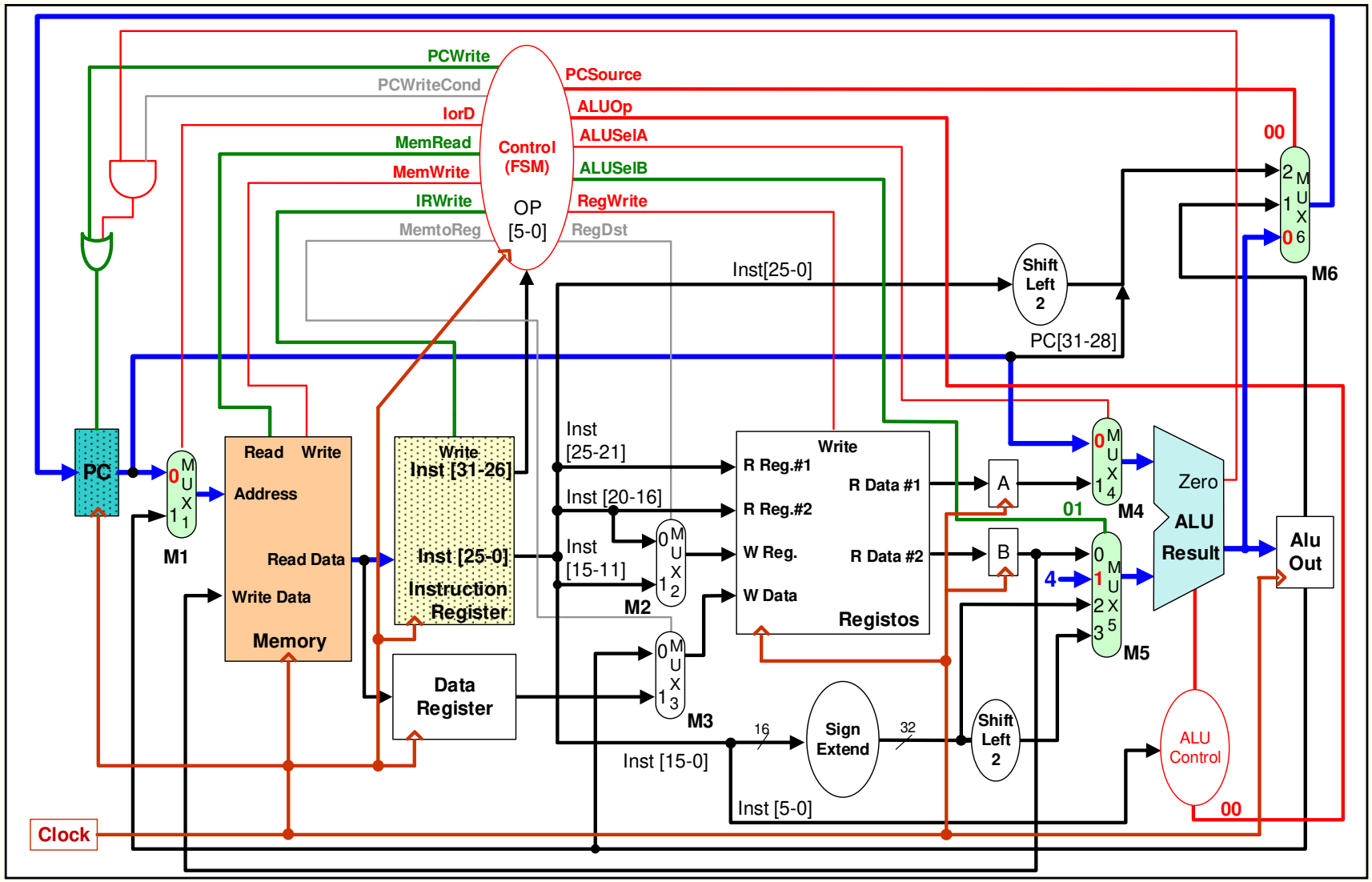
- Fase 1:
  - *Instruction fetch*
  - Cálculo de PC+4
- Fase 2:
  - Leitura dos registos
  - Descodificação da instrução
  - Cálculo do endereço-alvo das instruções de *branch*
- Fase 3:
  - Cálculo da operação na ALU
- Fase 4:
  - *Write-back*

Exemplo: *add $5,$8,$6*

Instruções do tipo R

Fase 1

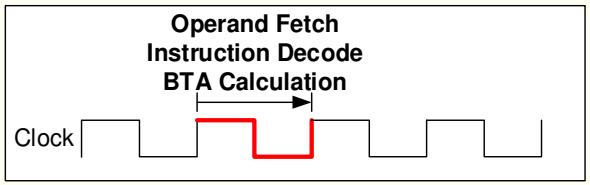


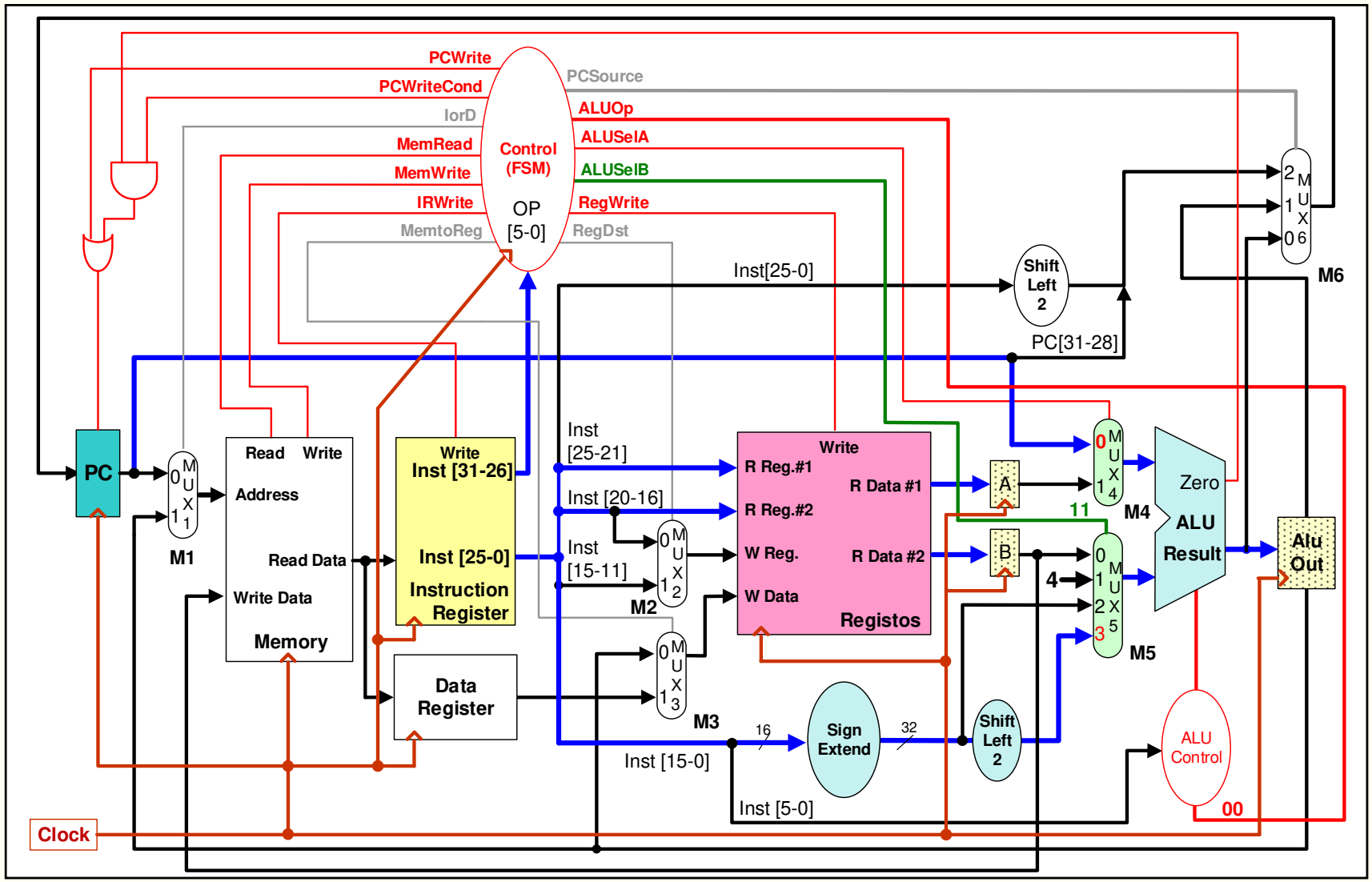


Ex.: add $5, $8, $6

Instruções do tipo R

Fase 2





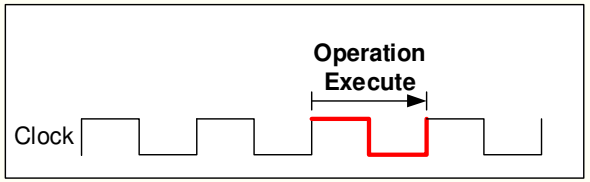
DETI-UA

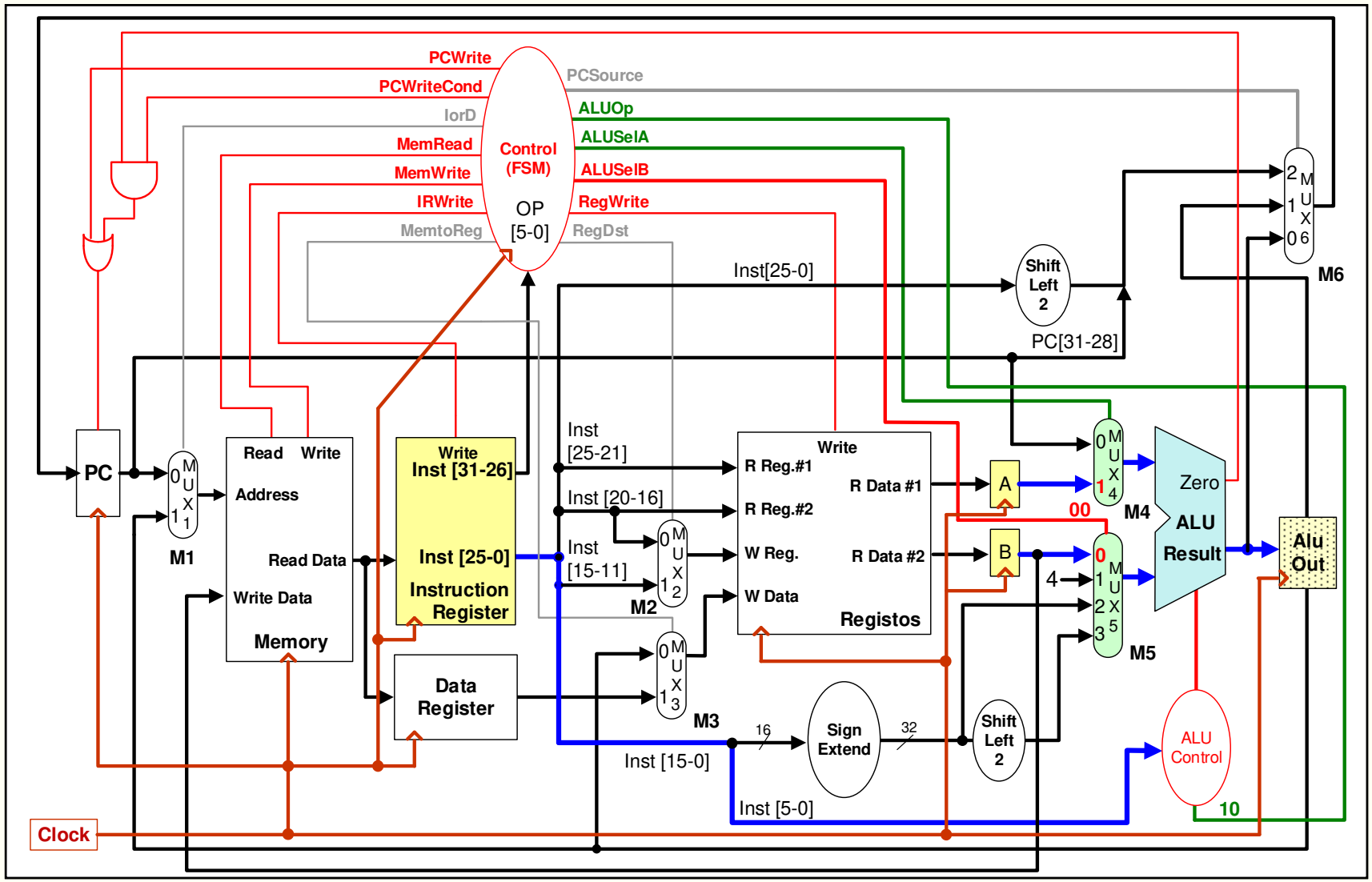
Ex.: add $5, $8, $6

Aulas 19 a 21 - 35

Instruções do tipo R

Fase 3





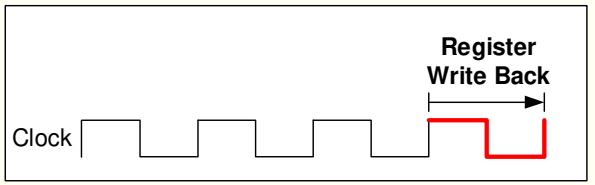
DETI-UA

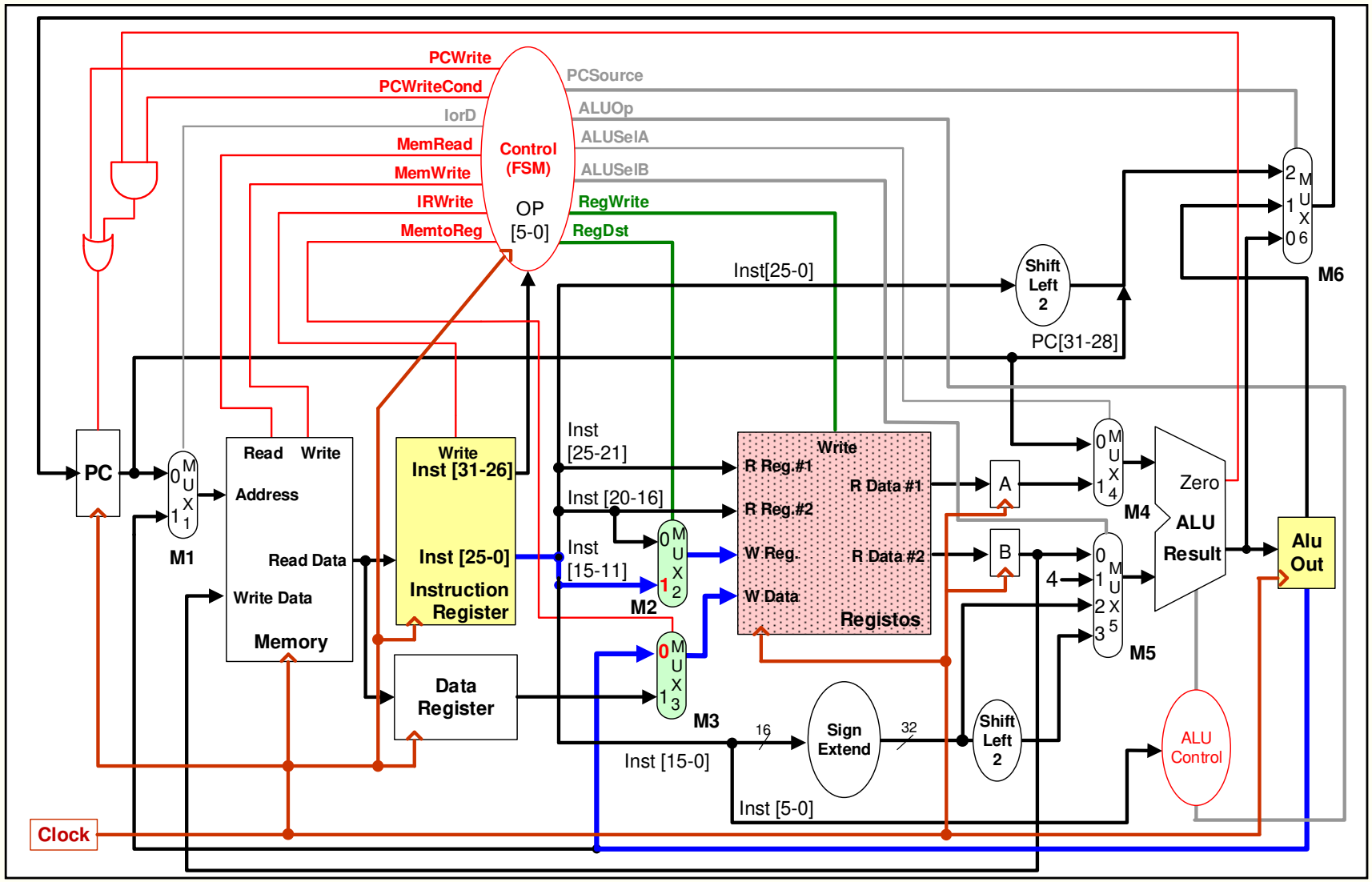
Ex.: add $5, $8, $6

Aulas 19 a 21 - 36

Instruções do tipo R

Fase 4





Ex.: add $5, $8, $6

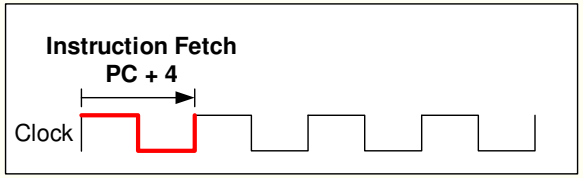
Funcionamento do *datapath* na instrução LW

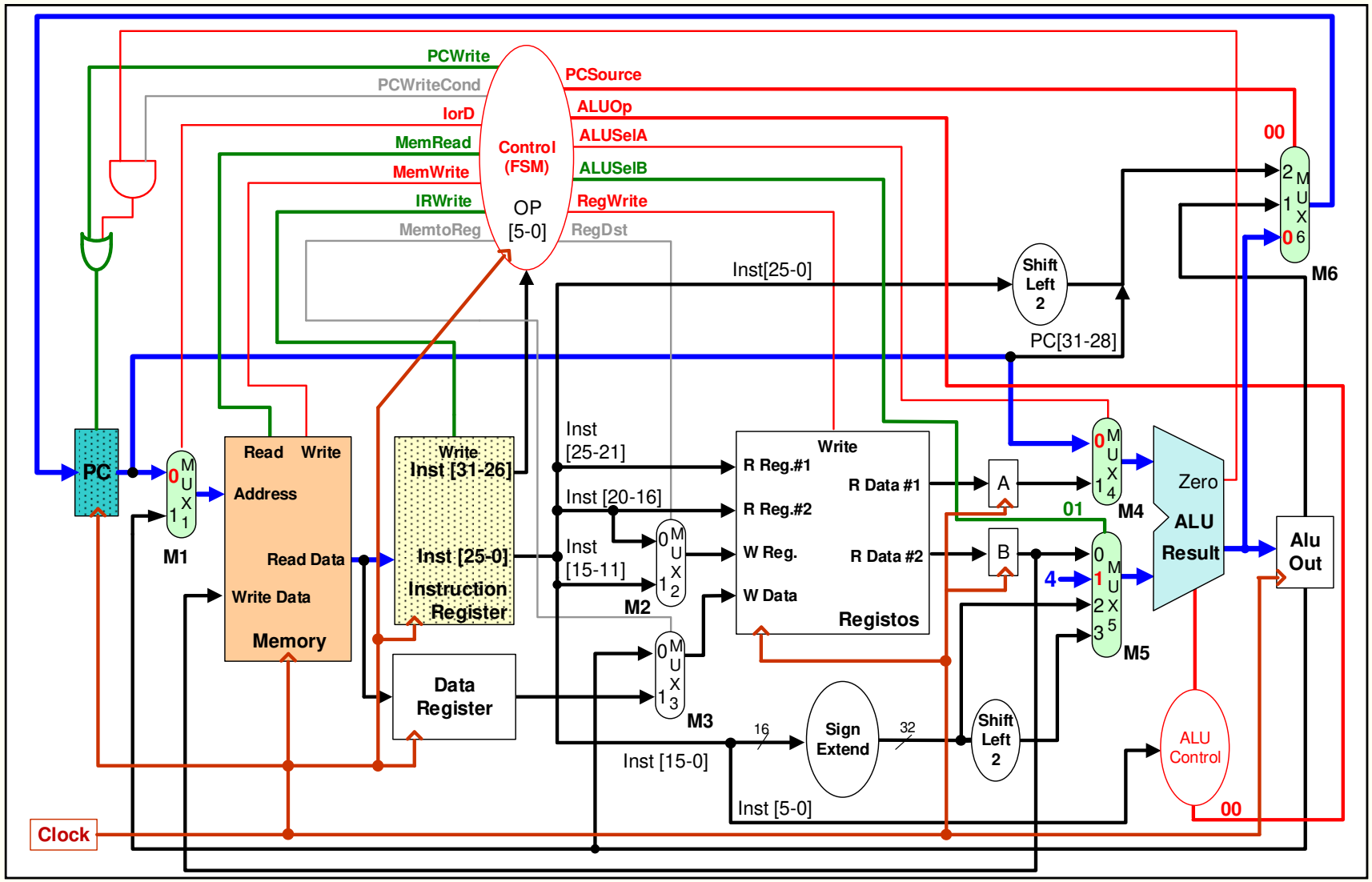
- Fase 1:
  - *Instruction fetch*
  - Cálculo de PC+4
- Fase 2:
  - Leitura dos registos
  - Descodificação da instrução
  - Cálculo do endereço-alvo das instruções de *branch*
- Fase 3:
  - Cálculo na ALU do endereço a aceder na memória
- Fase 4:
  - Leitura da memória
- Fase 5:
  - *Write-back*

***Exemplo**: lw $3,0x0014($6)*

Instrução LW

Fase 1

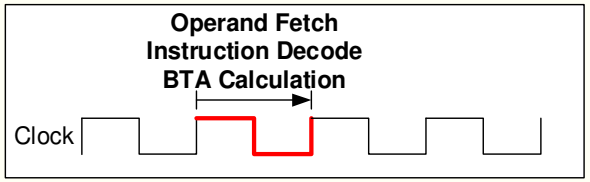


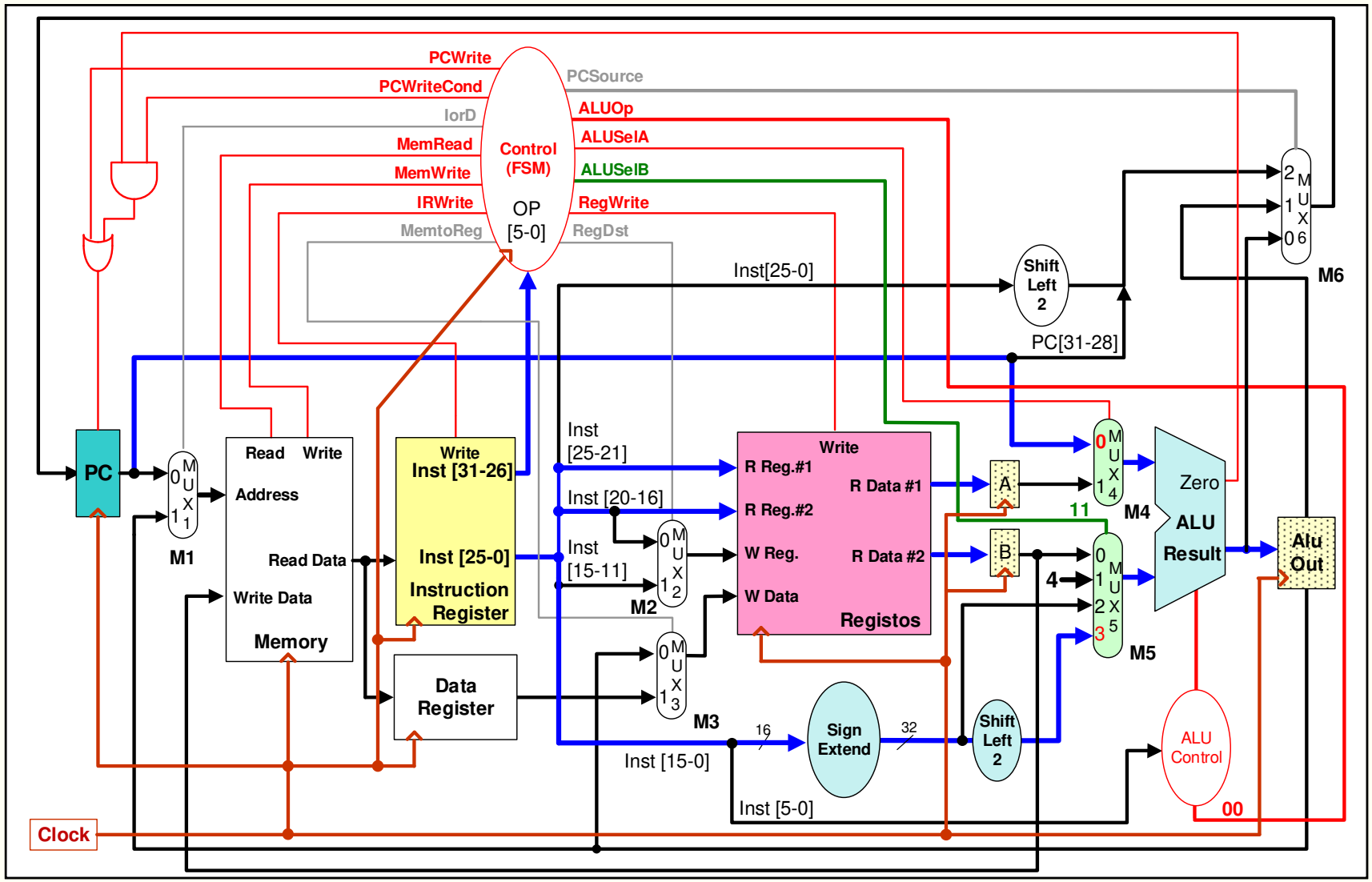


Ex.: lw $3,0x0014($6)

Instrução LW

Fase 2





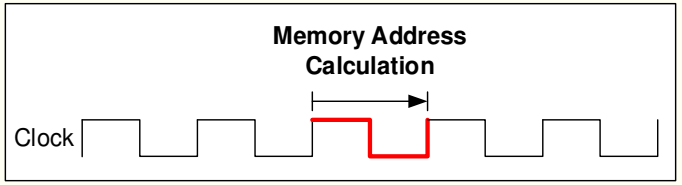
DETI-UA

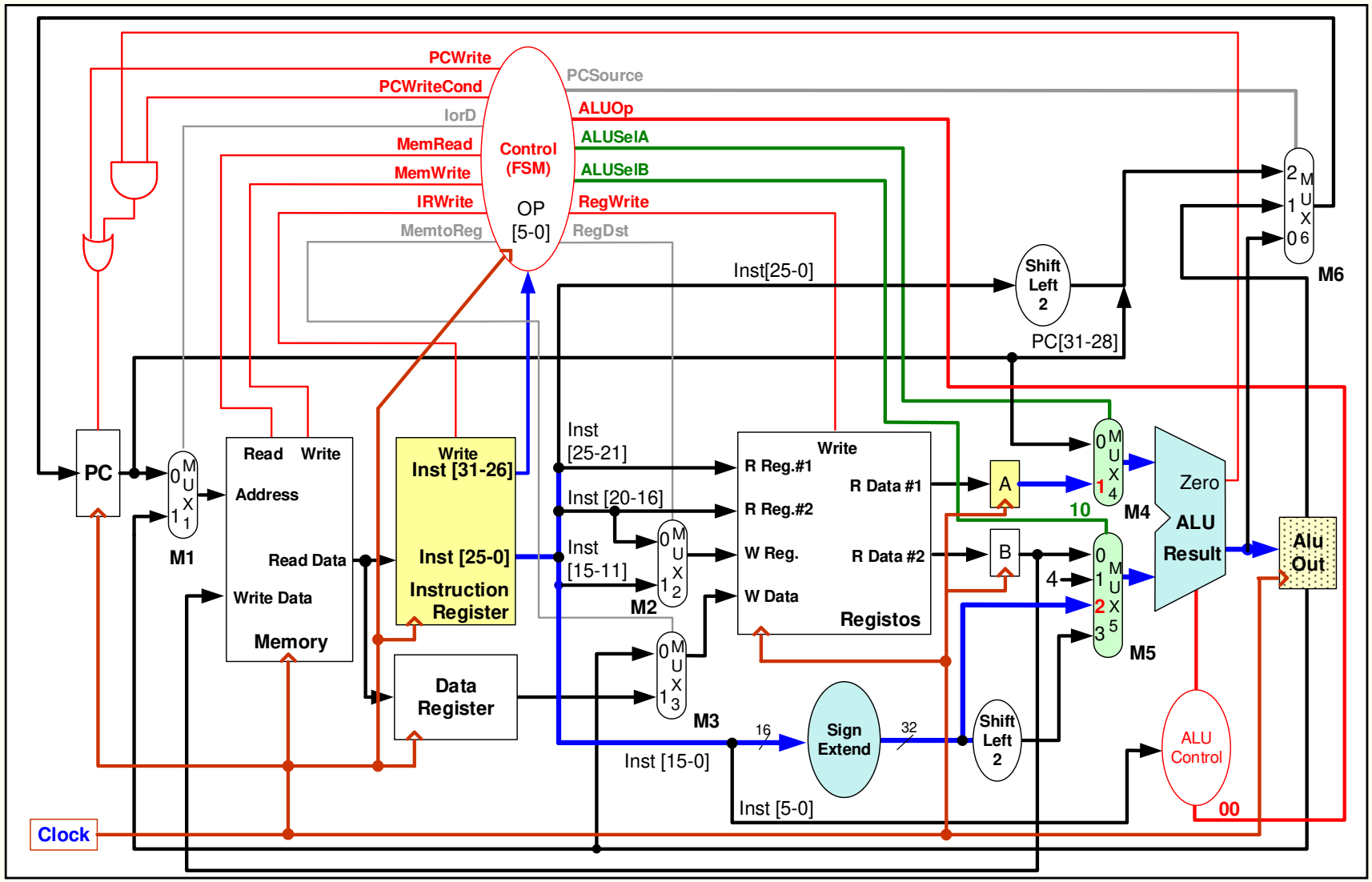
Ex.: lw $3,0x0014($6)

Aulas 19 a 21 - 40

Instrução LW

Fase 3





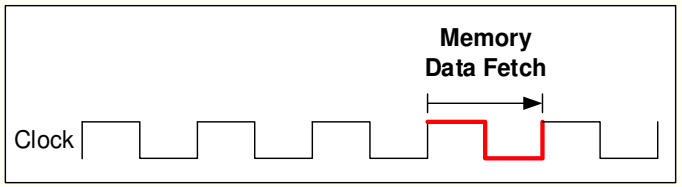
DETI-UA

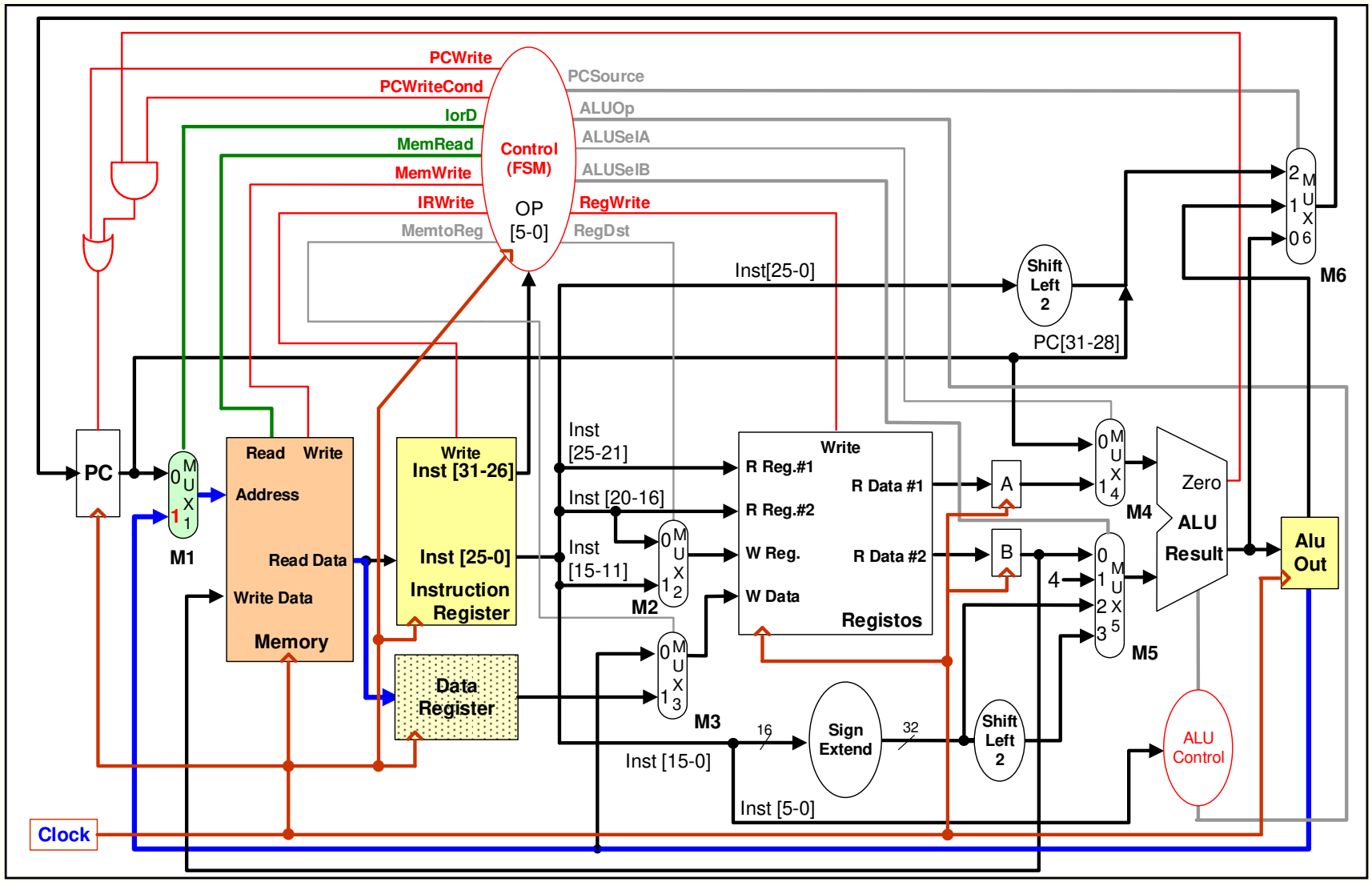
Ex.: lw $3,0x0014($6)

Aulas 19 a 21 - 41

Instrução LW

Fase 4

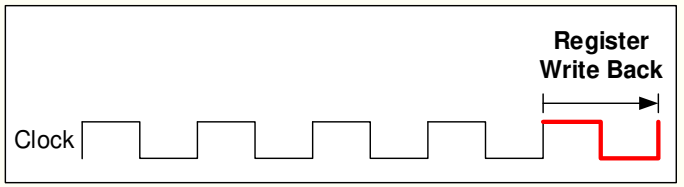


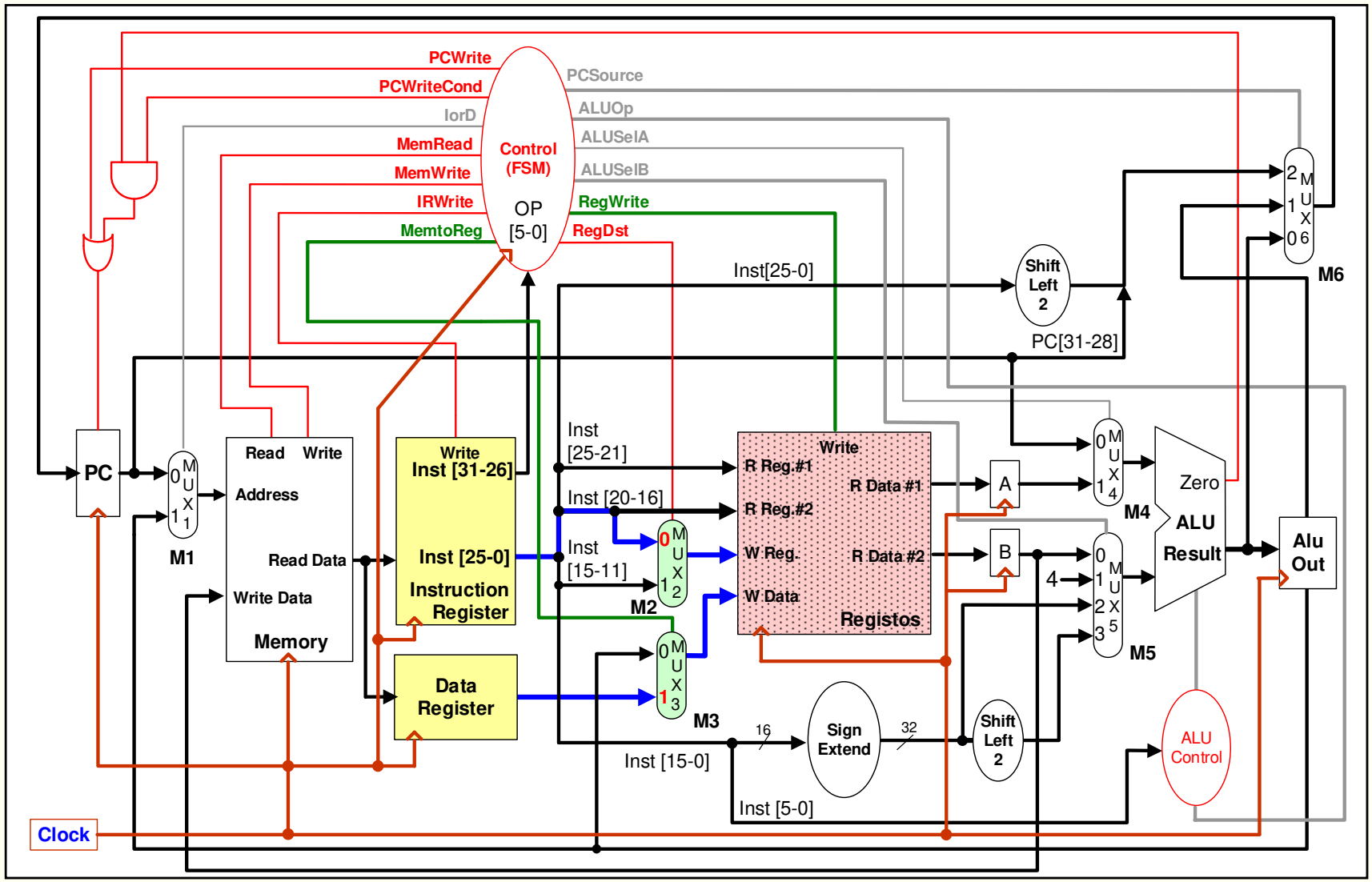


Ex.: lw $3,0x0014($6)

Instrução LW

Fase 5





Ex.: lw $3,0x0014($6)

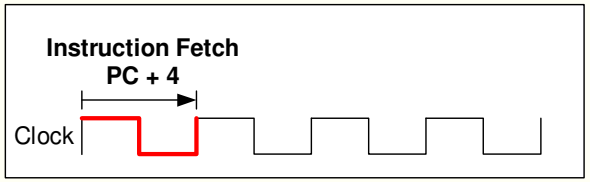
Funcionamento do *datapath* na instrução BEQ

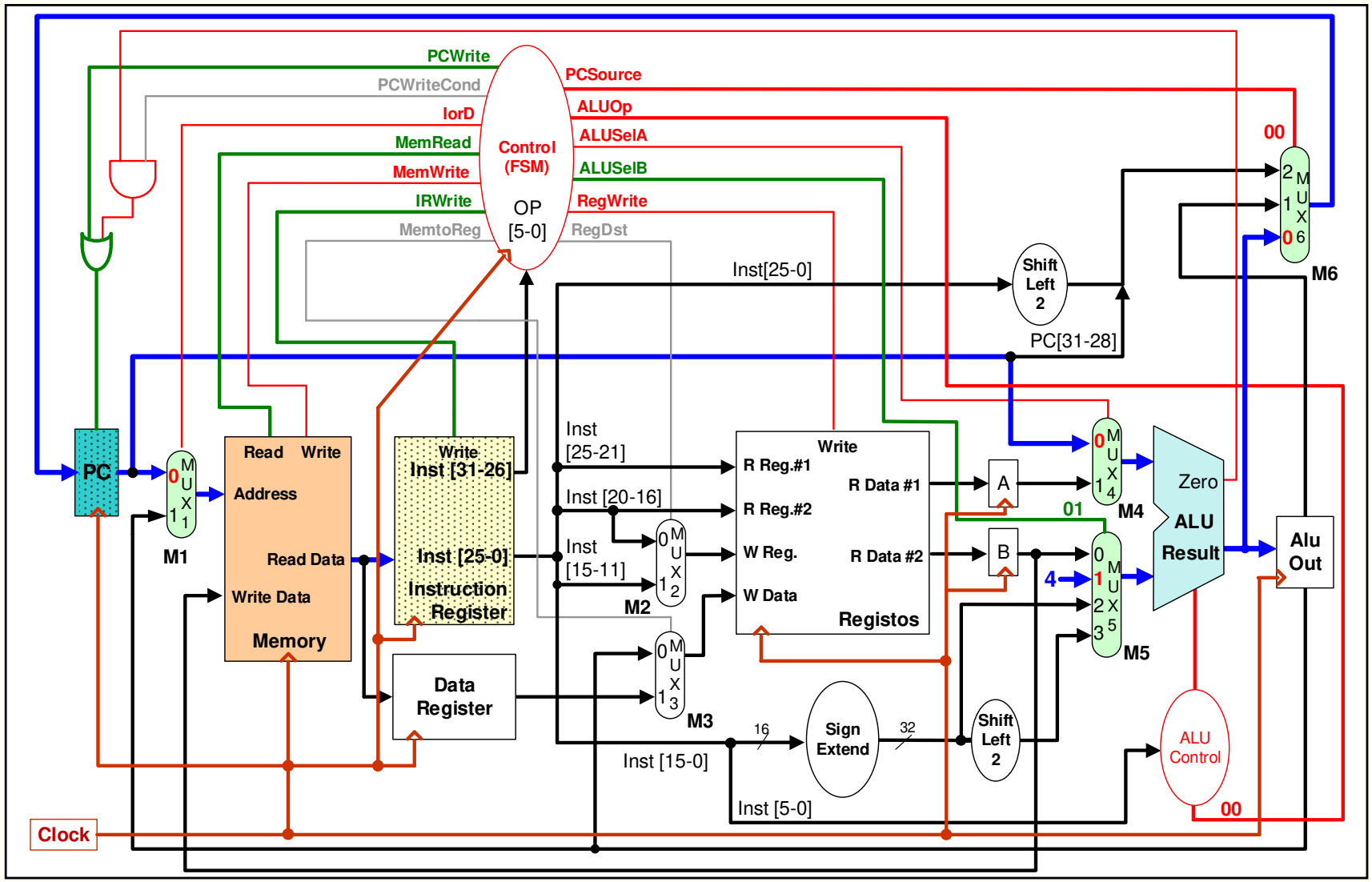
- Fase 1:
  - *Instruction fetch*
  - Cálculo de PC+4
- Fase 2:
  - Leitura dos registos
  - Descodificação da instrução
  - Cálculo do endereço-alvo das instruções de *branch* (BTA)
- Fase 3:
  - Comparação dos dois registos na ALU (subtração)
  - Conclusão da instrução de *branch* com eventual escrita do registo PC com o BTA

Exemplo: *beq $3,$6,endif*

Instrução BEQ

Fase 1





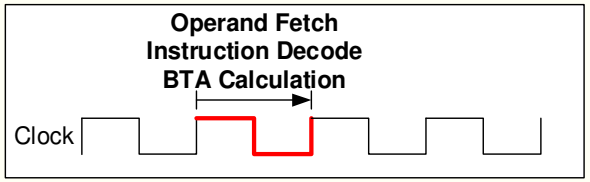
DETI-UA

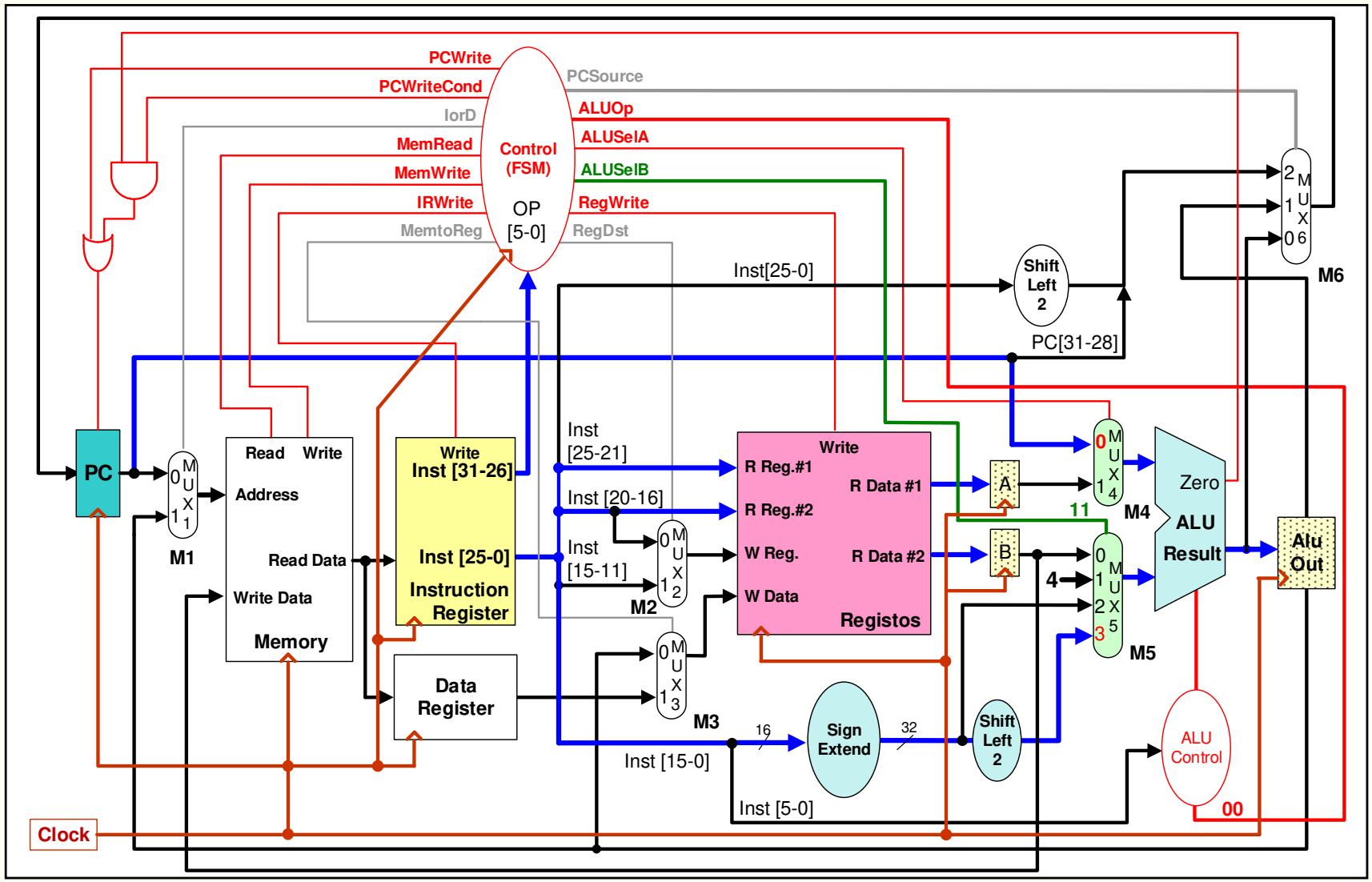
Ex.: beq $3, $6, endif

Aulas 19 a 21 - 45

Instrução BEQ

Fase 2





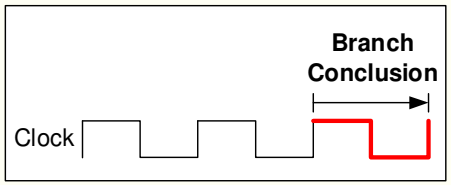
DETI-UA

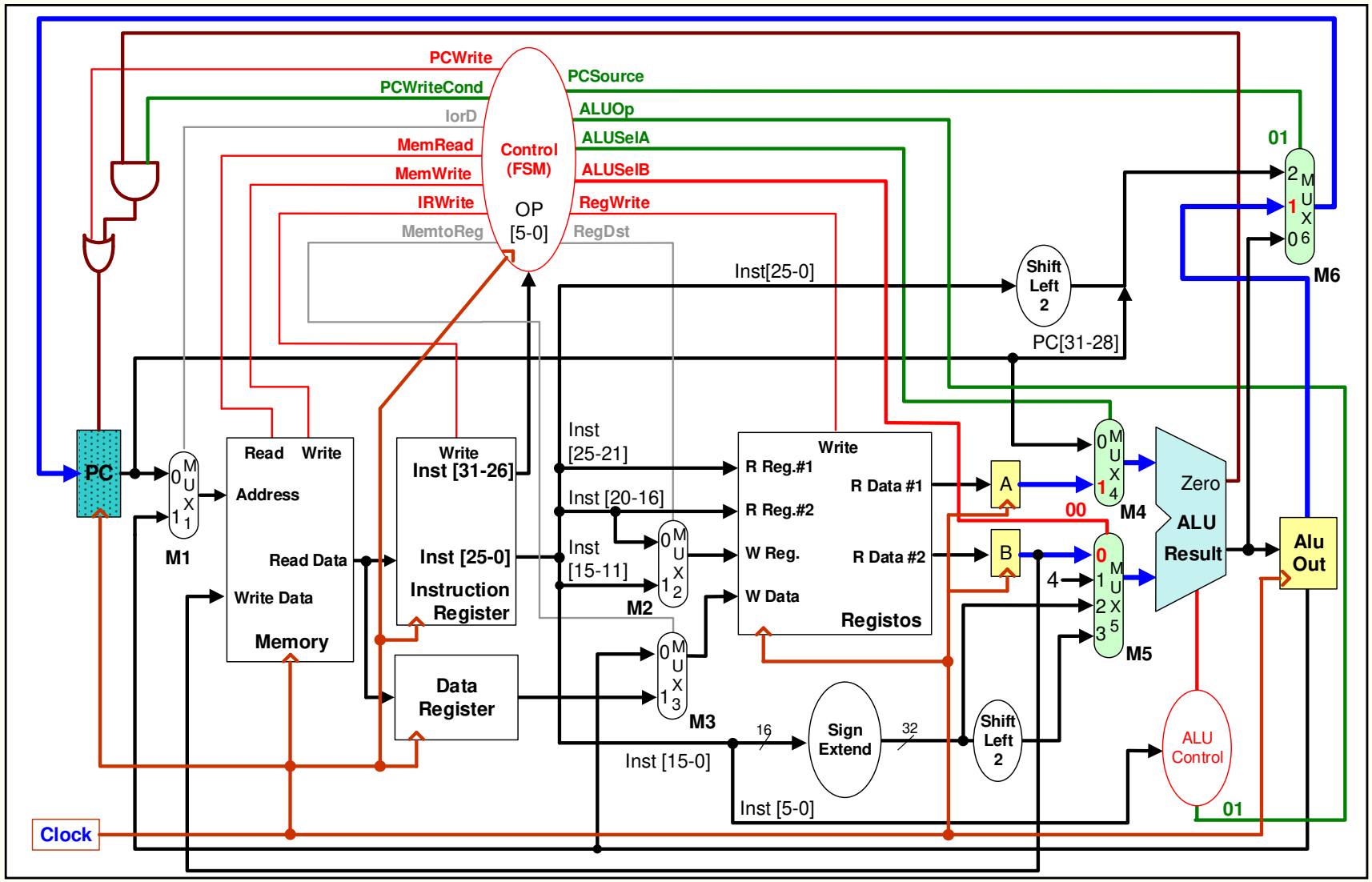
Ex.: beq $3, $6, endif

Aulas 19 a 21 - 46

Instrução BEQ

Fase 3





Ex.: beq $3, $6, endif

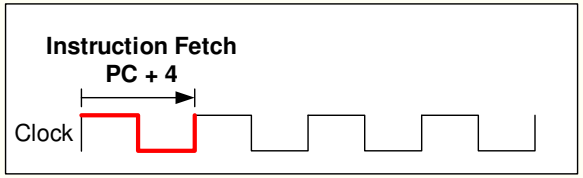
Funcionamento do *datapath* na instrução J

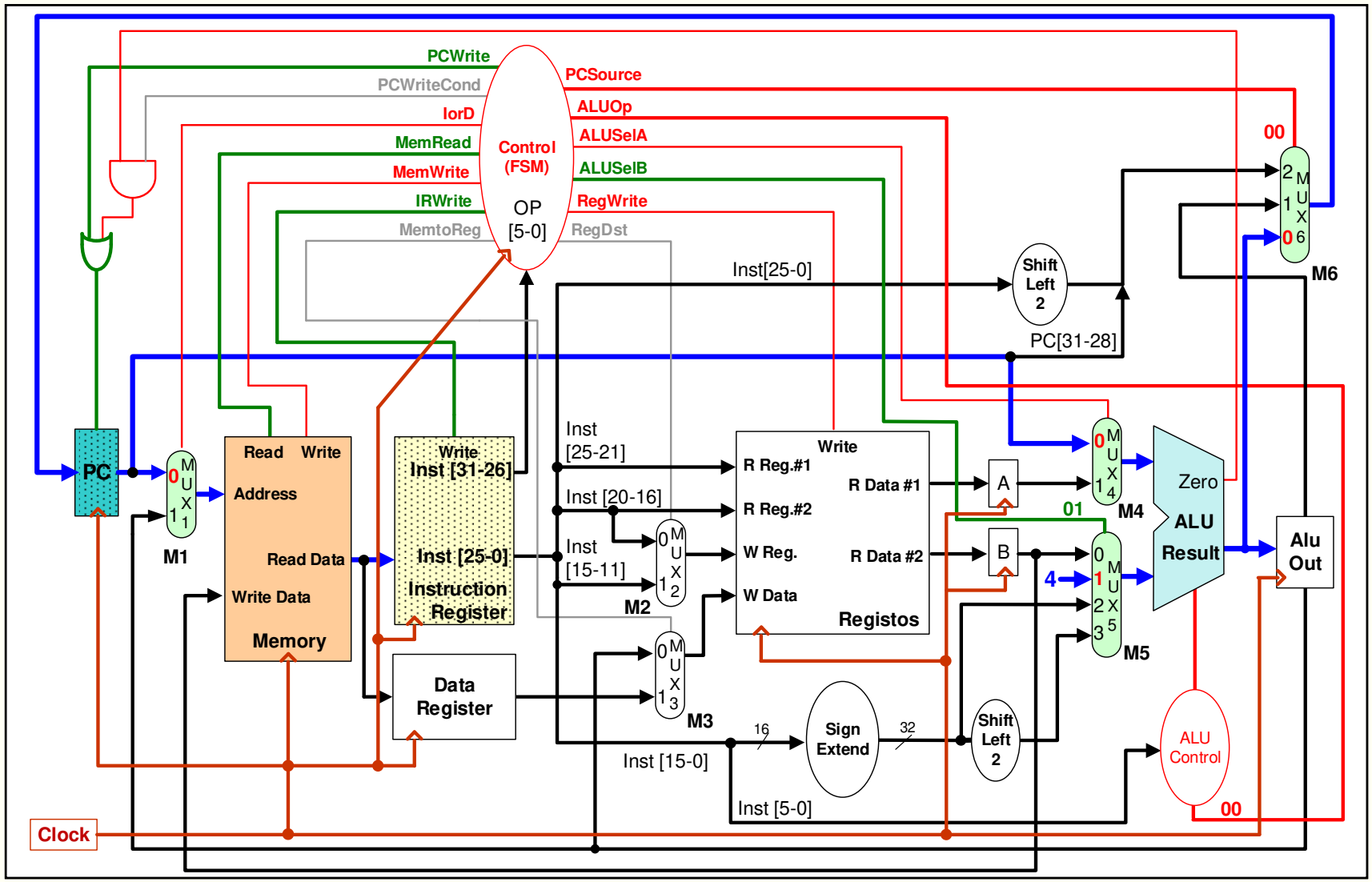
- Fase 1:
  - *Instruction fetch*
  - Cálculo de PC+4
- Fase 2:
  - Leitura dos registos
  - Descodificação da instrução
  - Cálculo do endereço-alvo das instruções de *branch*
- Fase 3:
  - Conclusão da instrução J com a seleção do JTA como próximo endereço do PC

Exemplo: *j loop*

Instrução J

Fase 1

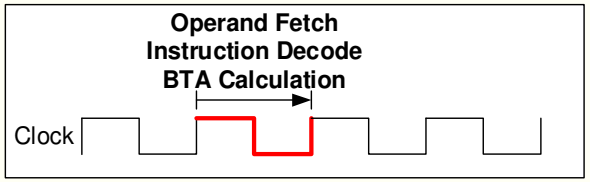


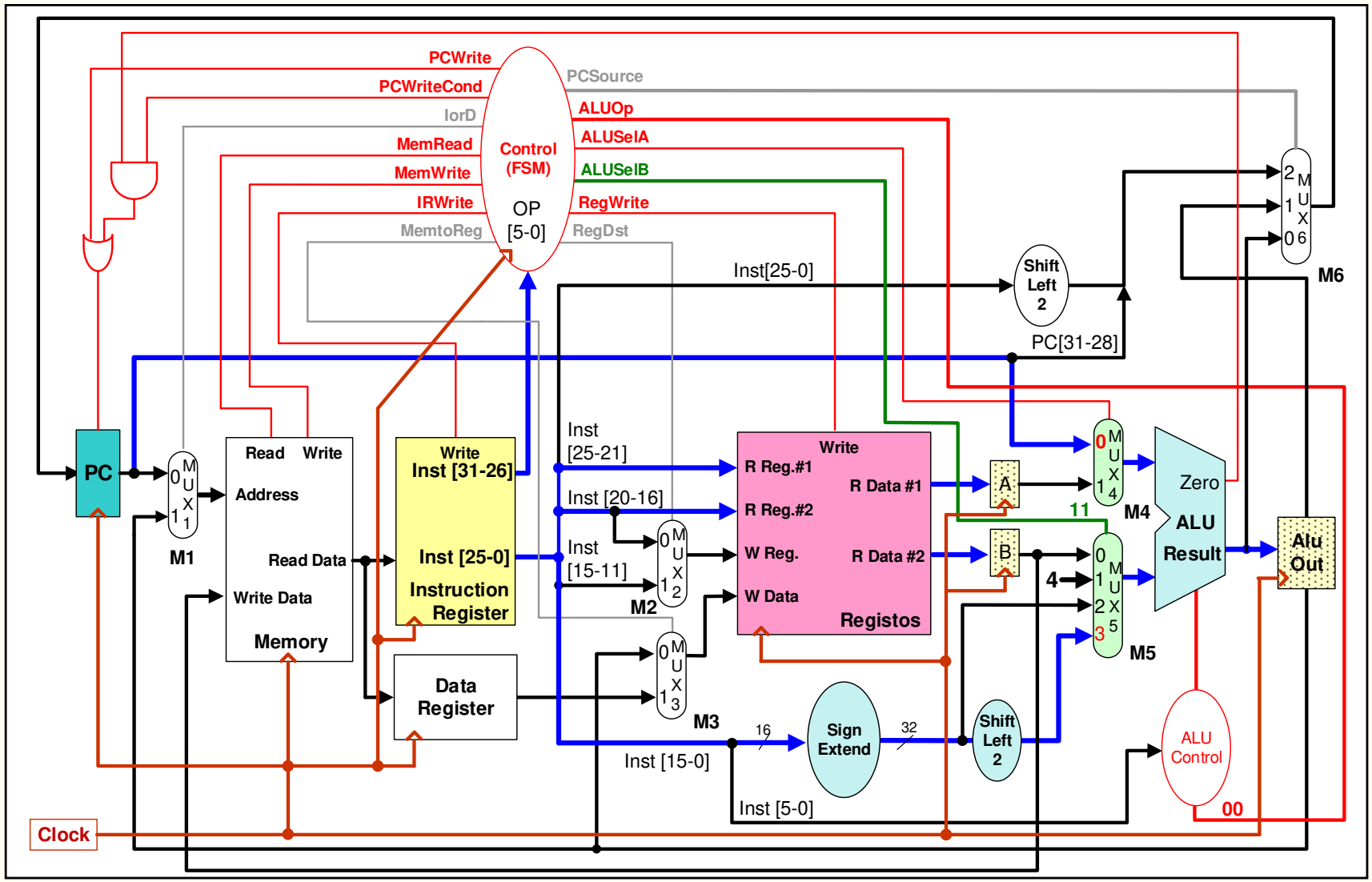


Ex.: j loop

Instrução J

Fase 2





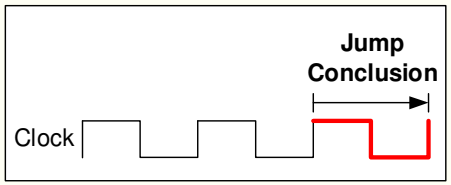
DETI-UA

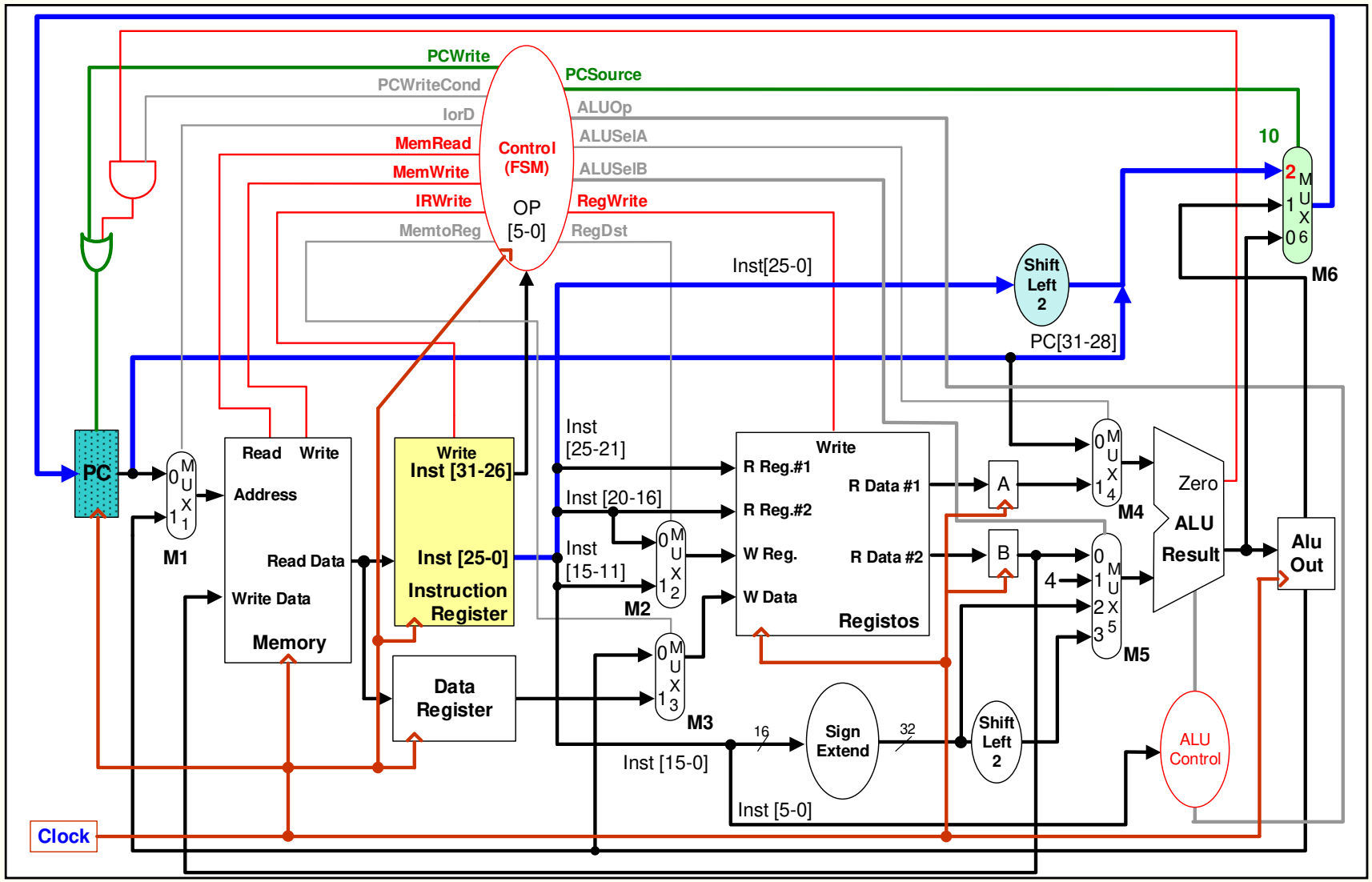
Ex.: j loop

Aulas 19 a 21 - 50

Instrução J

Fase 3



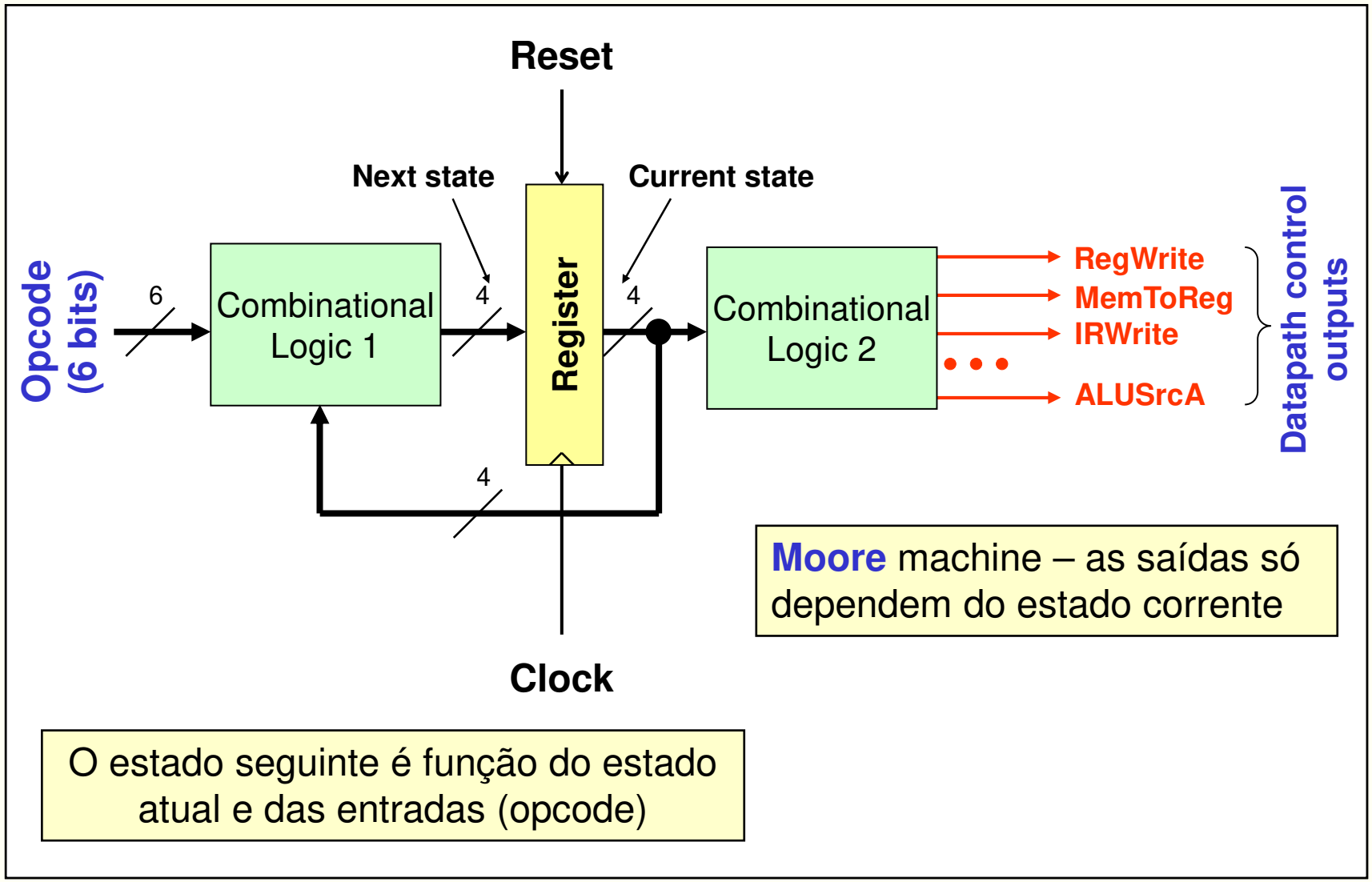


Ex.: j loop

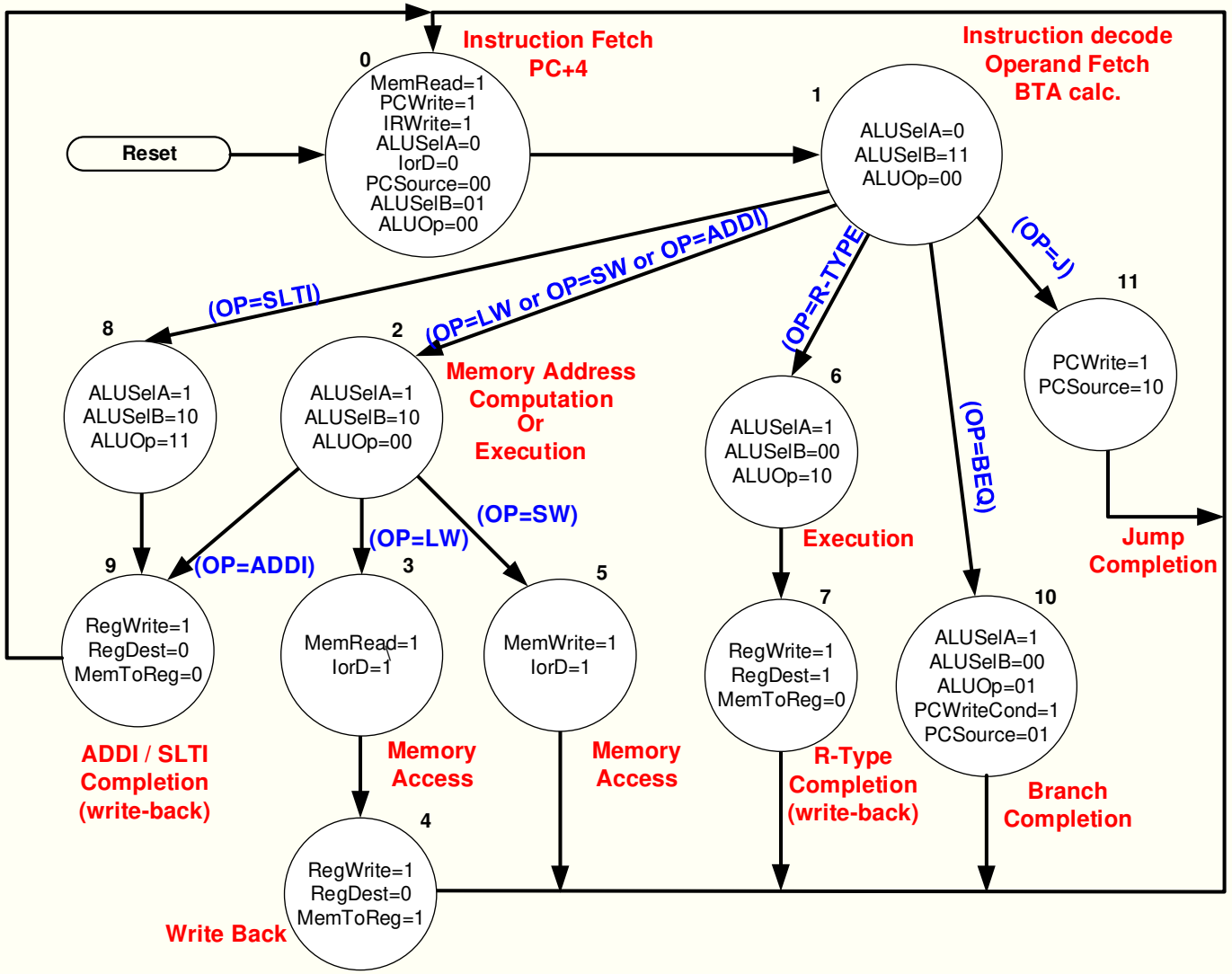
A unidade de controlo do *datapath Multi-cycle*

- No datapath ***single-cycle***, cada instrução é executada num único ciclo de relógio:
  - a unidade de controlo é responsável pela geração de um conjunto de sinais que não se alteram durante a execução de cada instrução.
  - a relação entre os sinais de controlo e o código de operação pode assim ser gerado por um circuito meramente combinatório.
- No datapath ***multi-cycle***, cada instrução é decomposta num conjunto de ciclos de execução, correspondendo cada um destes a um período de relógio distinto:
  - os sinais de controlo diferem de ciclo de relógio para ciclo de relógio e após o segundo ciclo diferem de instrução para instrução.
  - a solução combinatória deixa portanto de poder ser utilizada neste caso, sendo necessário recorrer a uma máquina de estados.

A unidade de controlo do *datapath Multi-cycle*



A unidade de controlo do *datapath Multi-cycle*



Os sinais de saída não explicitados em cada estado ou são irrelevantes (e.g. multiplexers) ou encontram-se no estado não ativo (controlo de elementos de estado)

A unidade de controlo do *datapath Multi-cycle* - VHDL

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

entity ControlUnit is
  port( Clock     : in std_logic;
        Reset     : in std_logic;
        OpCode    : in std_logic_vector(5 downto 0);
        PCWrite   : out std_logic;
        IRWrite   : out std_logic;
        IorD      : out std_logic;
        PCSource  : out std_logic_vector(1 downto 0);
        RegDest   : out std_logic;
        PCWriteCond : out std_logic;
        MemRead   : out std_logic;
        MemWrite  : out std_logic;
        MemToReg  : out std_logic;
        ALUSelA   : out std_logic;
        ALUSelB   : out std_logic_vector(1 downto 0);
        RegWrite  : out std_logic;
        ALUop     : out std_logic_vector(1 downto 0));
end ControlUnit;
```

A unidade de controlo do *datapath Multi-cycle* - VHDL

```
architecture Behavioral of ControlUnit is
  type TState is ( E0, E1, E2, E3, E4, E5, E6 , E7, E8, E9,
                   E10, E11);
  signal CS, NS : TState;
begin
  -- processo síncrono da máquina de estados (ME)
  process(Clock) is
  begin
     if(rising_edge(Clock)) then
        if(Reset = '1') then
           CS <= E0;
         else
           CS <= NS;
        end if;
     end if;
  end process;
 -- processo combinatório da ME na próxima página
end Behavioral;
```

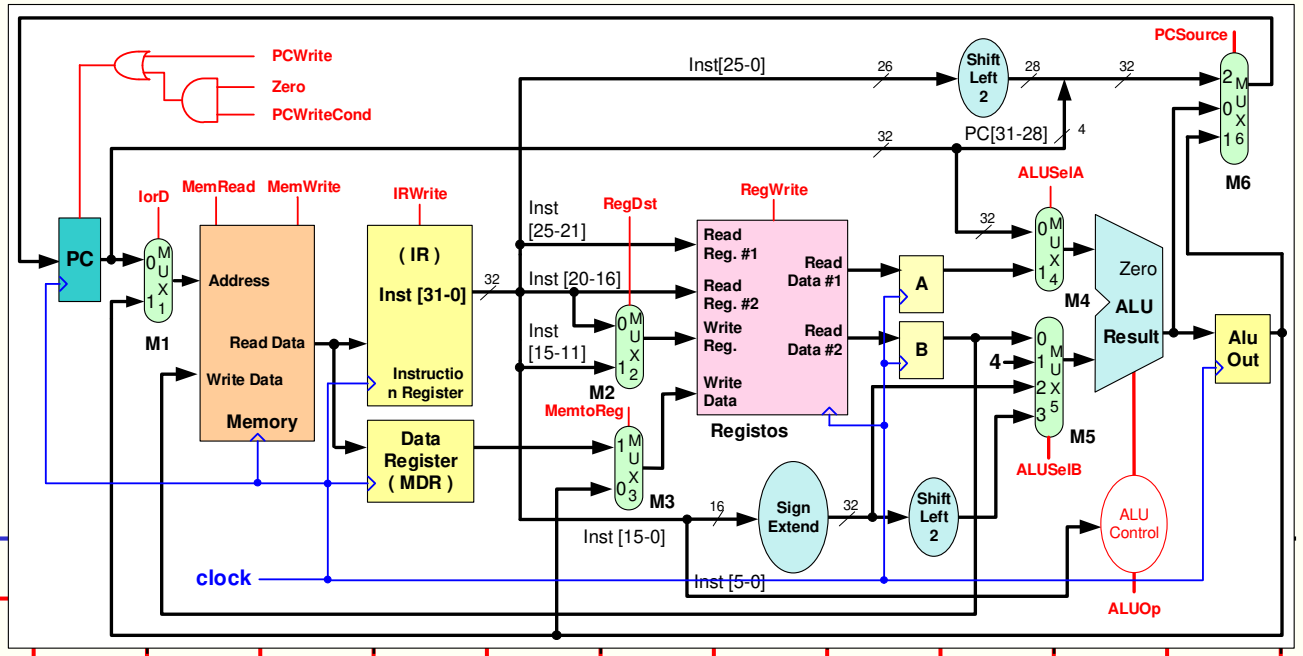
```
process(CS, OpCode) is
begin
    PCWrite<= '0'; IRWrite <= '0'; IorD <= '0'; RegDest <= '0';
    PCWriteCond<= '0'; MemRead <= '0'; MemWrite <= '0'; MemToReg <= '0';
    RegWrite <= '0';  PCSource <= "00"; ALUop <= "00"; ALUSelA <= '0';
    ALUSelB <= "00";
    NS <= CS;
    case CS is
        when E0 =>
            MemRead <= '1'; PCWrite <= '1'; IRWrite <= '1'; ALUSelB <= "01";
            NS <= E1;
        when E1 =>
            ALUSelB <= "11";
            if(OpCode = "000000") then NS <= E6;     -- R-Type instructions
            elsif(OpCode = "100011" or OpCode = "101011" or
                    OpCode = "001000") then          -- LW, SW, ADDI
                NS <= E2;
            elsif(OpCode = "001010") then NS <= E8; -- SLTI
            elsif(OpCode = "000100") then NS <= E10;-- BEQ
            elsif(OpCode = "000010") then NS <= E11;-- J
            end if;
        when E6 =>     -- R-Type instructions
            ALUSelA <= '1'; ALUop <= "10";
            NS <= E7;
        when E7 =>     -- R-Type instructions
            RegWrite <= '1';  RegDest <= '1';
            NS <= E0;
        -- (...)
    end case;
end process;
```

Processo combinatório

Funcionamento do DP

add	$2, $3, $4
sw	**$2, -4($6)**
or	$4, $6, $3

Sinais de controlo na execução sequencial das três instruções:



add $2, $3, $4

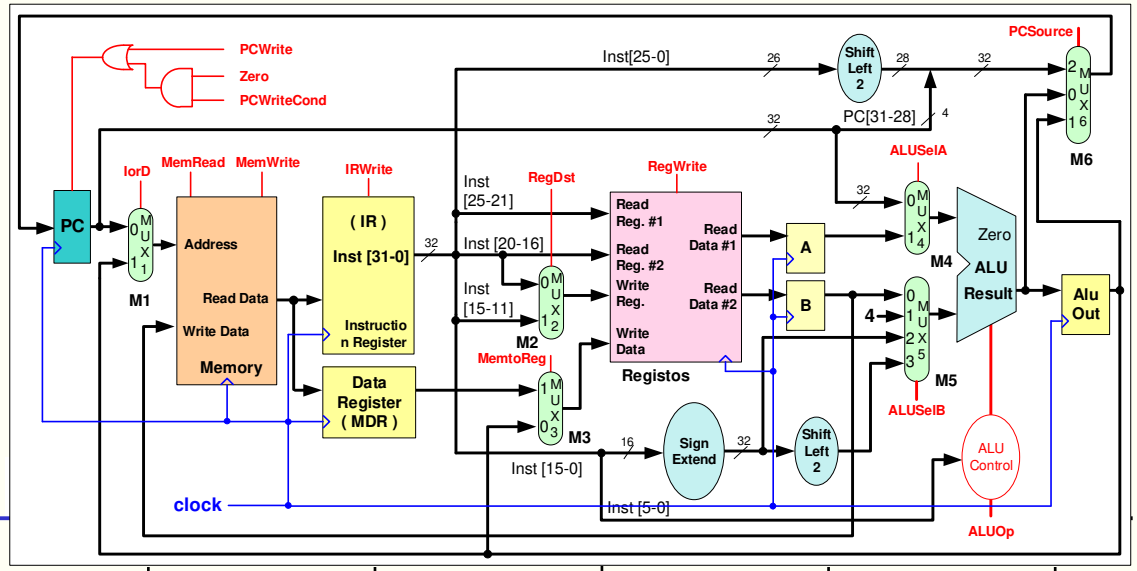
PCWriteCond	0	X	0	0	0	X	0
PCWrite	0	1	0	0	0	1	0
MemWrite	0	0	0	0	1	0	0
MemRead	0	1	0	0	0	1	0
MemToReg	0	X	X	X	X	X	X
IRWrite	0	1	0	0	0	1	0
ALUSelA	X	0	0	1	X	0	0
ALUSelB	XX	01	11	10	XX	01	11
ALUOp	XX	00	00	00	XX	00	00
IorD	X	0	X	X	1	0	X
PCSource	XX	00	XX	XX	XX	00	XX
RegWrite	1	0	0	0	0	0	0
RegDst	1	X	X	X	X	X	X

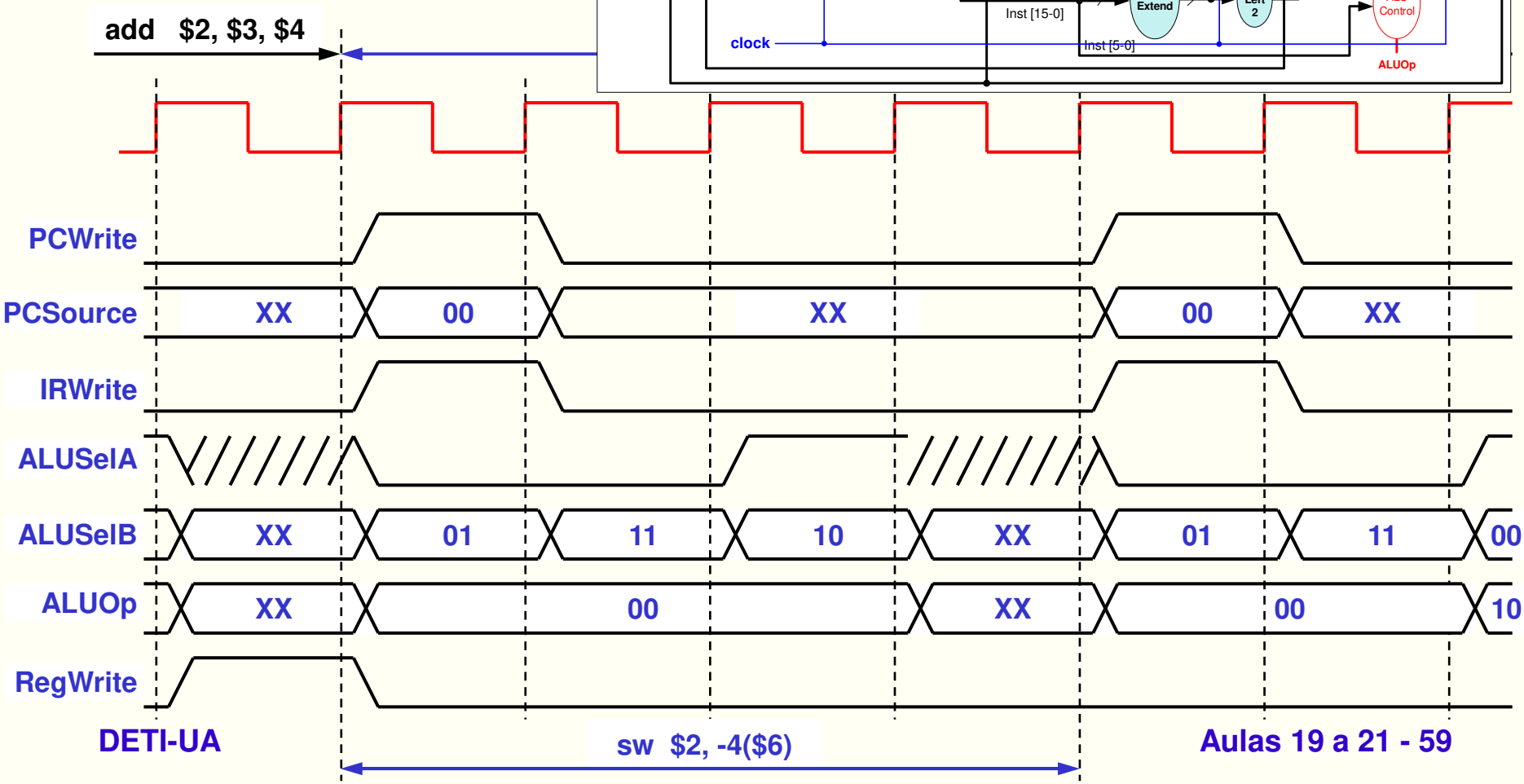
sw $2, -4($6)

Funcionamento do DP

add	$2, $3, $4
sw	**$2, -4($6)**
or	$4, $6, $3

Sinais de controlo: diagrama temporal





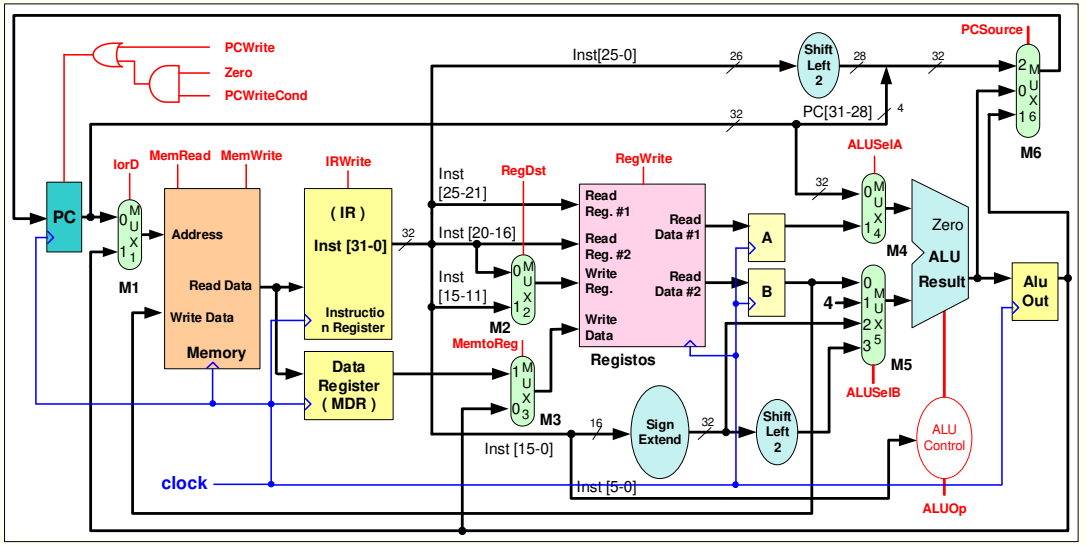
Funcionamento do DP

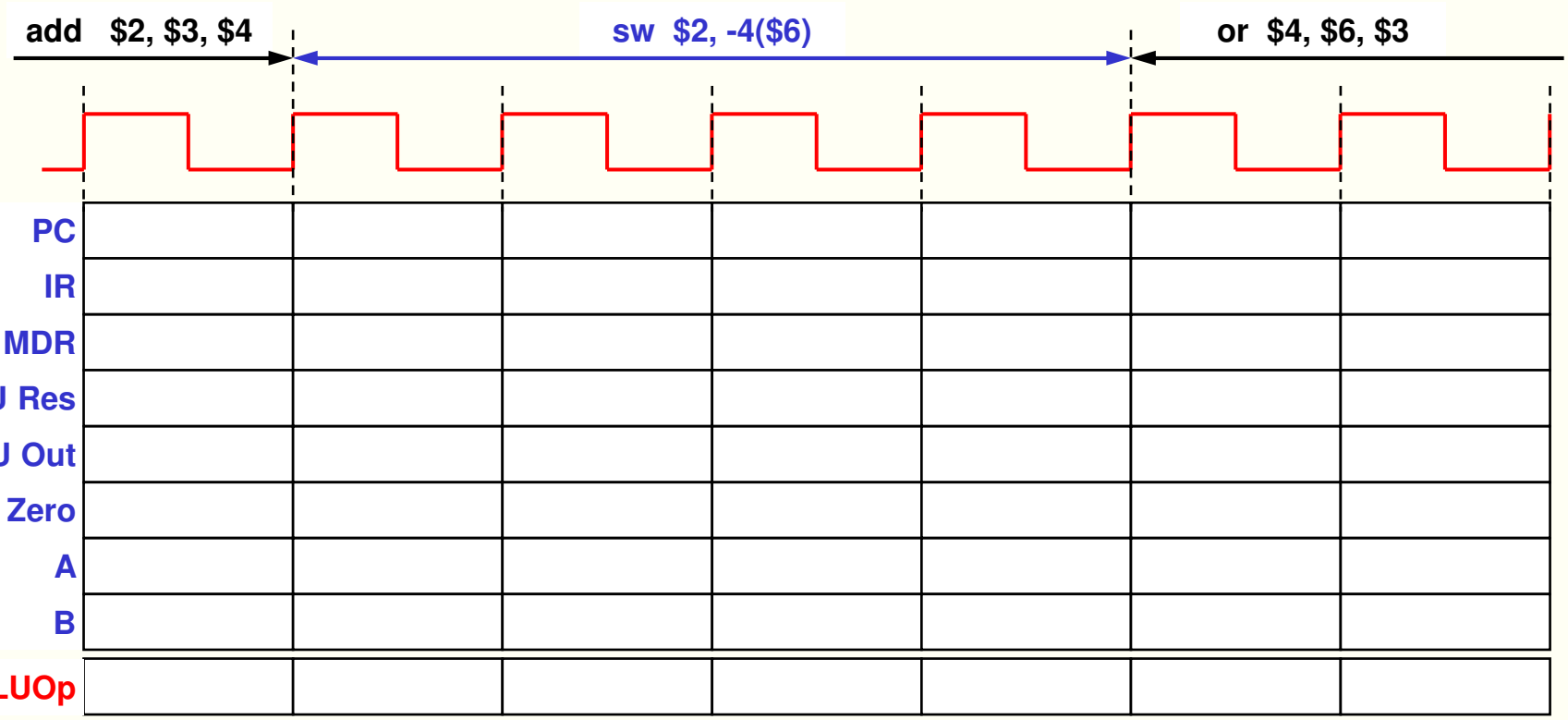
```
00400048  add $2, $3, $4   # 00641020
0040004C  sw  $2, -4($6)   # ACC2FFFC
00400050  or  $4, $6, $3   # 00C32025
```

(valores em hexadecimal)

$3	20001FA6
$4	81002378
$6	10012480

Valores calculados / obtidos em cada ciclo de relógio:





Opcodes: **SW** - 0x2B, **ADD** - 0x20, **OR** - 0x25

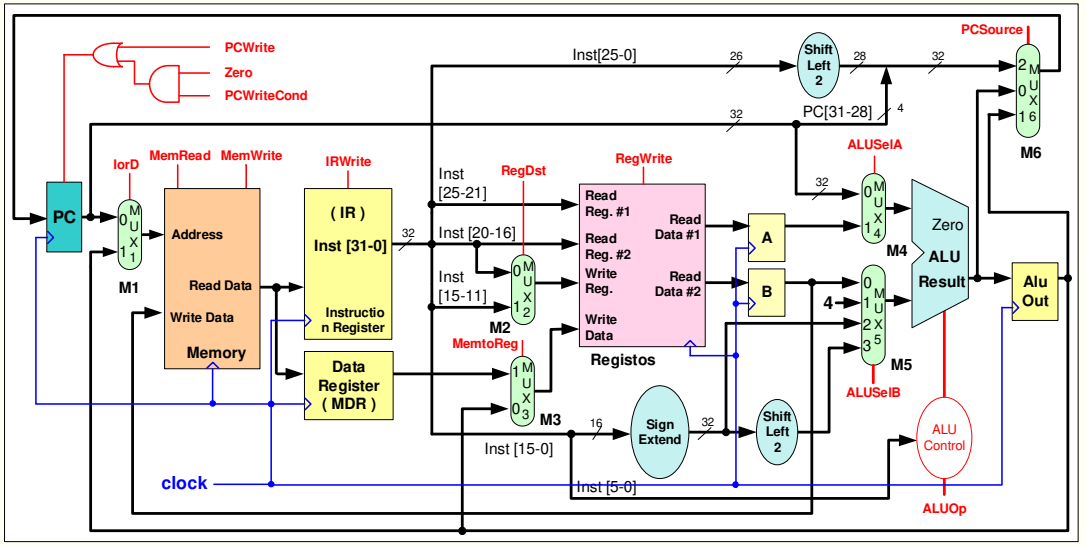
Funcionamento do DP

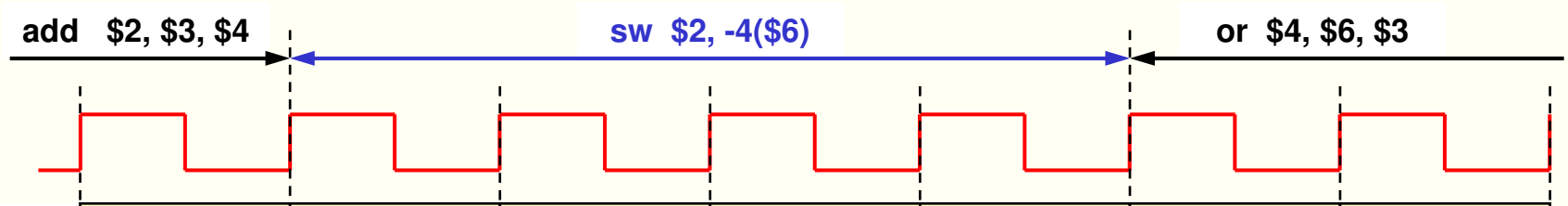
```
00400048  add $2, $3, $4   # 00641020
0040004C  sw  $2, -4($6)   # ACC2FFFC
00400050  or  $4, $6, $3   # 00C32025
```

(valores em hexadecimal)

$3	20001FA6
$4	81002378
$6	10012480

Valores calculados / obtidos em cada ciclo de relógio:





PC	0040004C	0040004C	00400050	00400050	00400050	00400050	00400054
IR	00641020	00641020	ACC2FFFC	ACC2FFFC	ACC2FFFC	ACC2FFFC	00C32025
MDR	?	?	ACC2FFFC	?	?	?	00C32025
ALU Res	?	00400050	00400040	1001247C	?	00400054	004080E8
ALU Out	A100431E	?	00400050	00400040	1001247C	?	00400054
ALU Zero	?	0	0	0	?	0	0
A	20001FA6	20001FA6	20001FA6	10012480	10012480	10012480	10012480
B	81002378	81002378	81002378	A100431E	A100431E	A100431E	A100431E
ALUOp	XX	00	00	00	XX	00	00

Opcodes: SW - 0x2B, ADD - 0x20, OR - 0x25

Exercícios

- Considere um programa que executa em 10s num computador "A" com uma frequência de 4GHz. Pretende-se desenvolver um computador "B" que execute o programa em 6s. O hardware *designer* verificou que é possível um aumento da frequência de trabalho do CPU do computador "B", mas isso acarreta um acréscimo do número total de ciclos de relógio de 1,2 vezes relativamente a A. Qual a frequência de trabalho que deverá ter o CPU da máquina "B"?
- Considere duas máquinas com implementações distintas da mesma arquitetura do conjunto de instruções (ISA). Para um dado programa,
  - Máquina A: Clock_cycle = 350 ps; CPI = 2,0
  - Máquina B: Clock_cycle = 400 ps; CPI = 1,5

  Qual a máquina mais rápida? Qual a relação de desempenho?
- Considere duas máquinas ("A" e "B") com implementações distintas da mesma arquitetura do conjunto de instruções (ISA). Para um mesmo programa, a máquina "A" apresenta um CPI de 2,0 e a "B" de 3,125. Usando a métrica tempo de execução, verificou-se que a máquina "A" é mais rápida que a máquina "B" por um fator de 1,25. Calcule a relação entre as frequências de relógio das máquinas "A" e "B".

Exercícios

- Considerando os seguintes tempos de atraso dos elementos operativos do *datapath single-cycle* que estudou:
  - acesso à memória para leitura: 5ns; acesso à memória para preparar a escrita: 2ns; acesso ao *register file* para leitura: 3ns; acesso ao *register file* para preparar a escrita: 2ns; operação da ALU: 4ns; operação de um somador: 2ns; unidade de controlo: 2ns; tempo de *setup* do PC: 1ns; extensor de sinal: 1ns; *left shifter*: 1ns; multiplexers: 0ns;
  - Q1: calcule o tempo mínimo de execução para cada uma das instruções suportadas.
  - Q2: calcule a frequência máxima de funcionamento do *datapath single-cycle.*
- O que limita a frequência máxima do relógio do *datapath multi-cycle*?

S

Exercícios

- Quantos ciclos de relógio demora, no *datapath multi-cycle*, a execução de cada uma das instruções consideradas (r-type, lw, sw, addi, slti, beq e j)?
- Para os tempos de atraso apresentados no exercício anterior, qual a frequência máxima de funcionamento do *datapath multi-cycle*?
- Considere um programa com 100.000 instruções, com o seguinte padrão: 10% de lw, 10% de sw, 60% de tipo R, 10% de addi/slti, 5% de branches e 5% de jumps. Usando os valores de frequência que calculou anteriormente, determine o tempo de execução desse programa: a) num *datapath single-cycle*; b) num *datapath multi-cycle*. Calcule, para esse programa, o ganho de desempenho da arquitetura *multi-cycle* relativamente à arquitetura *single-cycle*.

Exercícios

- Calcule o número de ciclos de relógio que o programa seguinte demora a executar, desde o *Instruction Fetch* da 1ª instrução até à conclusão da última instrução, tendo em atenção os valores da memória de dados apresentados:

  1) num *datapath single-cycle, 2)* num *datapath multi-cycle*

```
main:
        lw      $1,0($0)
        add     $4,$0,$0
        lw      $2,4($0)
loop:
        lw      $3,0($1)
        add     $4,$4,$3
        sw      $4,36($1)
        addi    $1,$1,4
        slt     $5,$1,$2
        bne     $5,$0,loop
        sw      $4,8($0)
        lw      $1,12($0)
```

Memória de dados	
Address	Value
0x0000000	0x10
0x0000004	0x20

Exercícios

- Calcule o número de ciclos de relógio que o programa seguinte demora a executar, desde o *Instruction Fetch* da 1ª instrução até à conclusão da última instrução, tendo em atenção os valores da memória de dados apresentados:

  1) num *datapath single-cycle, 2)* num *datapath multi-cycle*

```
main:                       # p0 = 0;
      lw     $1,0($0)       # p1 = *p0 = 0x10;
      add    $4,$0,$0       # v = 0;
      lw     $2,4($0)       # p2=*(p0+1)=0x20;
loop:                       # do {
      lw     $3,0($1)       #   aux1 = *p1;
      add    $4,$4,$3       #   v = v + *p1;
      sw     $4,36($1)      #   *(p1 + 9) = v;
      addi   $1,$1,4        #    p1++;
      slt    $5,$1,$2       #
      bne    $5,$0,loop     # } while(p1 < p2);
      sw     $4,8($0)       # *(p0 + 2) = v;
      lw     $1,12($0)      # aux2 = *(p0 + 3);
```

Memória de dados	
Address	Value
0x0000000	0x10
0x0000004	0x20

Exercícios

- Suponha que no endereço de memória **0x00400038** está armazenada a instrução "**lw $5,-12($7)**"; considere ainda que o conteúdo dos registos **$5** e **$7** é, respetivamente, **0x10013CA4** e **0x10010098**. Calcule os valores que estão disponíveis à saída do registo "**ALUOut**" durante as 2ª, 3ª e 4ª fases de execução dessa instrução.
- Preencha as tabelas dos slides 57 e 59 para a execução da instrução "**xor $10,$3,$17**", supondo que está armazenada no endereço **0x004000A0** e que o valor dos registos é: **$10=0xF3A431**, **$3=0xA1234**, **$17=0xFF0C8**.
- Complete o código VHDL da unidade de controlo apresentado nos slides 55 e 56 para todas as instruções definidas.