Implementação de um Subconjunto Multi-Ciclo do Processador MIPS

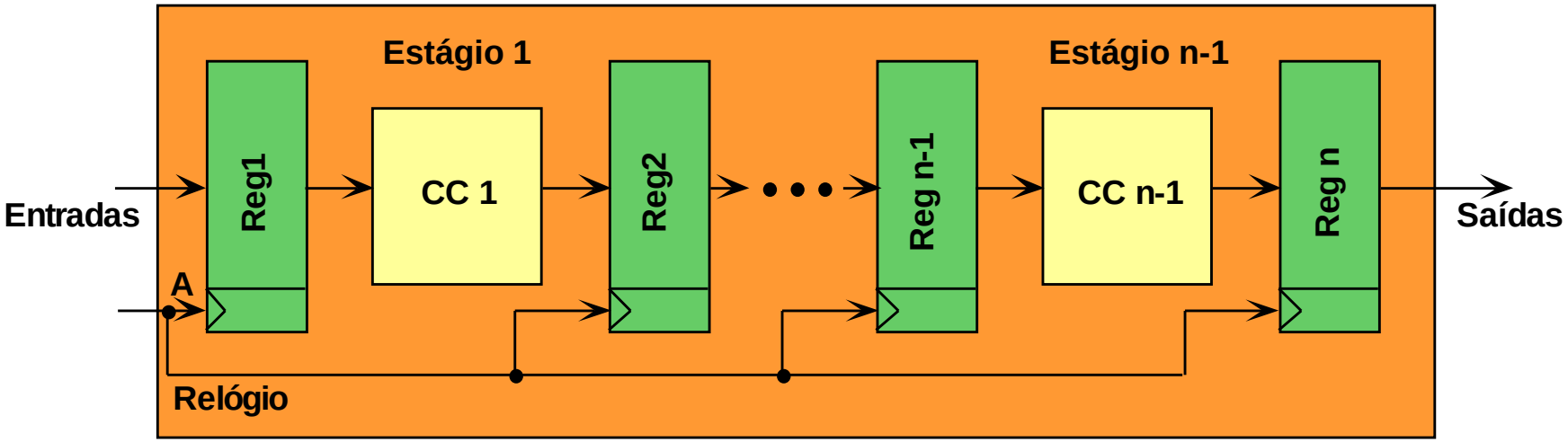
Fernando Moraes

09/10/2006

Última alteração - Ney Calazans, 23/11/2016

DESCRIÇÃO RTL de um HW MULTI-CICLO

- **Cada estágio realiza uma parte do trabalho**
- **Registradores usados para isolar os resultados – são as chamadas “barreiras temporais”**

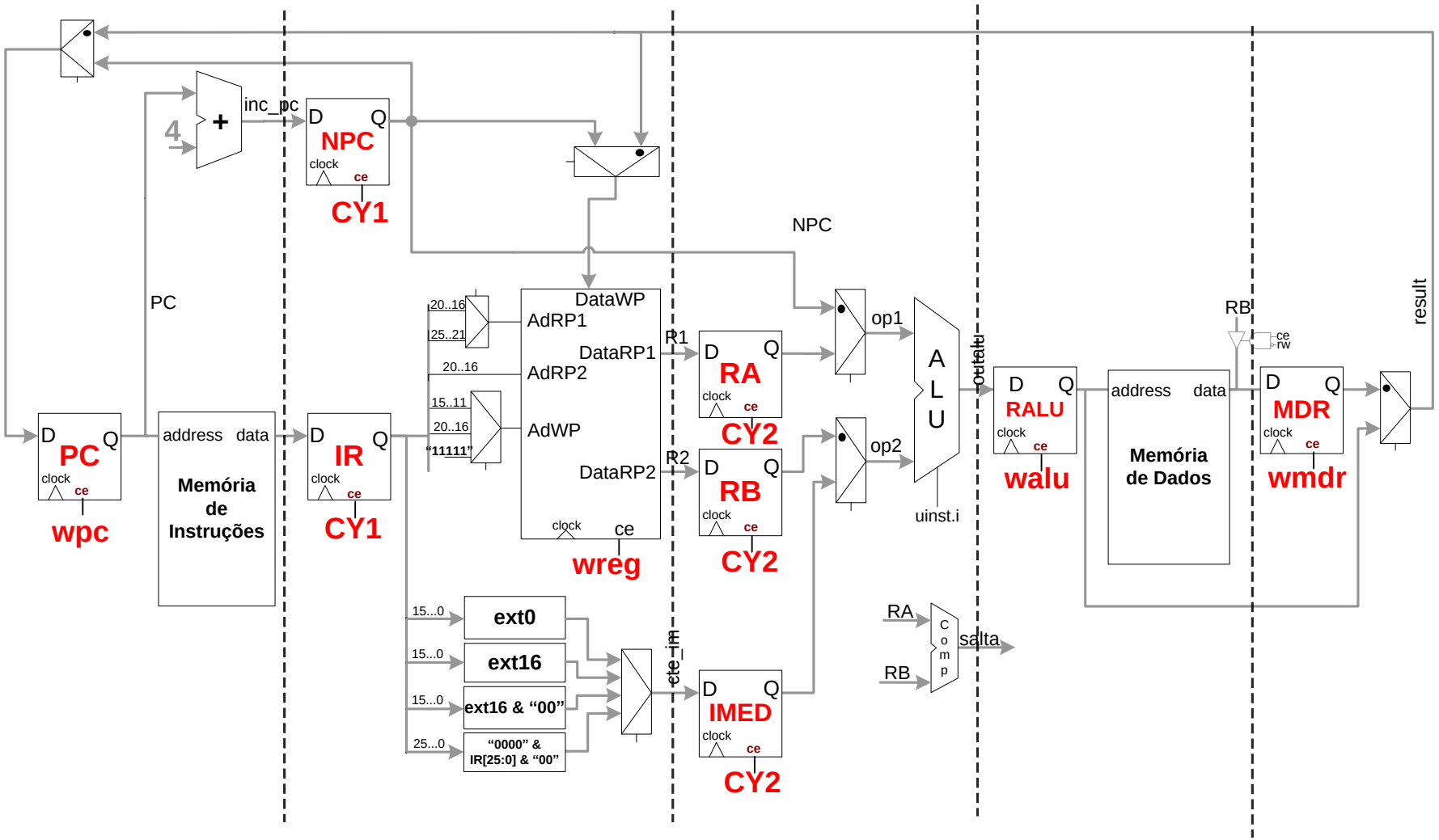


Especificação da Organização MIPS-MC

- **Dá suporte a 37 das 153 instruções da ISA MIPS-32™**
  - **Aritméticas (5): ADDU, SUBU, MULTU, DIVU, ADDIU**
  - **Lógicas (7): AND, OR, XOR, NOR, ANDI, ORI, XORI**
  - **Deslocamento de bits (6): SLL, SLLV, SRA, SRAV, SRL, SRLV**
  - **Acesso à Memória (4): LBU, LW, SB, SW**
  - **Teste (4): SLT, SLTU, SLTI, SLTIU**
  - **Controle de Fluxo (8): BEQ, BGEZ, BLEZ, BNE, J, JAL, JALR, JR**
  - **Miscelâneas (3): MFHI, MFLO, LUI**
- **Comparar com as 9 instruções da MIPS monociclo**
- **Maior parte das instruções executa em 4 ciclos de relógio**
- **LW e LBU executam em 5 ciclos**
- **MULTU e DIVU executam em 67 ciclos**

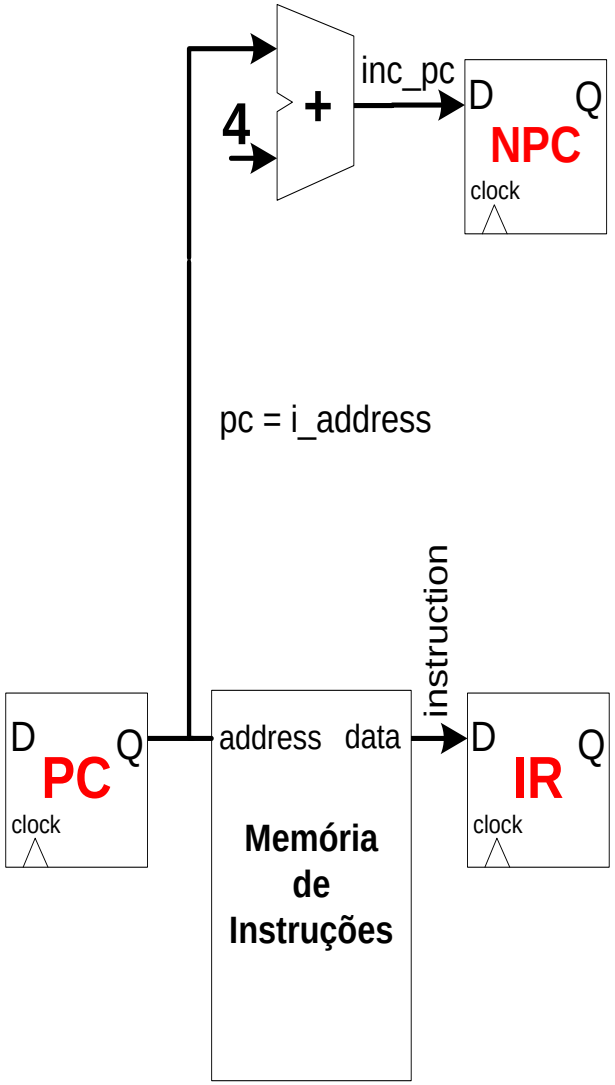
Estrutura Geral com Sinais de Controle

- **Obs: Estrutura não considera Hw para MULTU/DIVU**



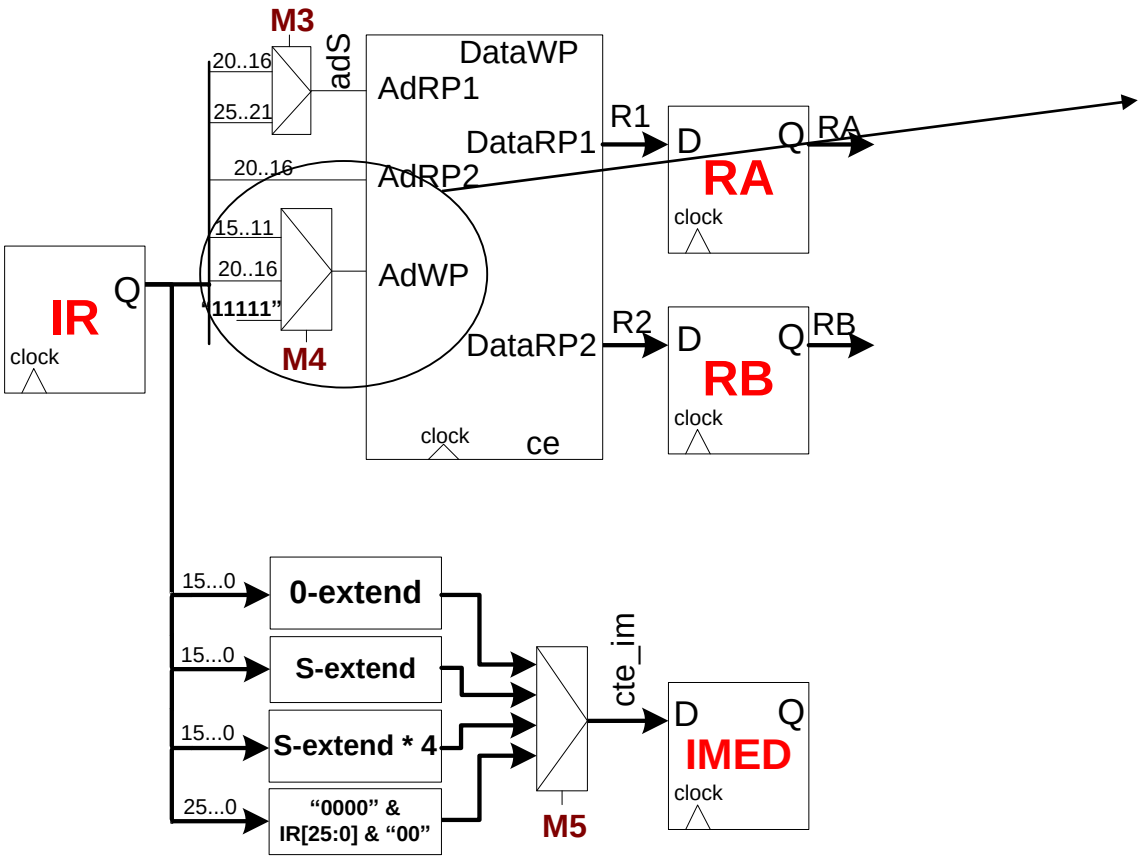
Primeiro estágio

- **Comum a todas as instruções**
  - **Incrementa o PC**
  - **Armazena instrução no IR**
  - **NPC é novo registrador**
    - Porquê existe?
      - Um motivo: não gerar transitórios em saltos
      - Ver especificação das instruções!



Segundo estágio

- **Comum para todas as instruções**
- **Leitura do Banco de Registradores**
- **Determinação da constante imediata**



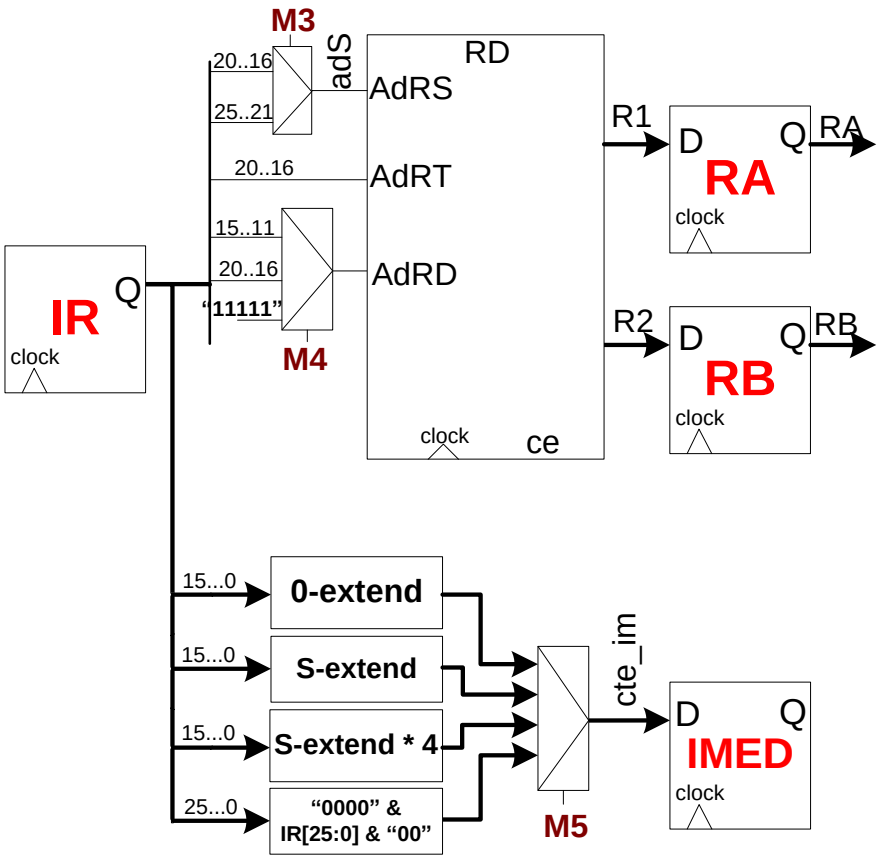
Controle de escrita no banco de registradores.

Discutido no último estágio

Segundo estágio

- **Controle do multiplexador M3**
  - **Determina o endereço do primeiro registrador a ser lido**

```
adS <= IR(20 downto 16) when uins.i=SSLL or uins.i=SSRA or uins.i=SSRL else
       IR(25 downto 21);
```

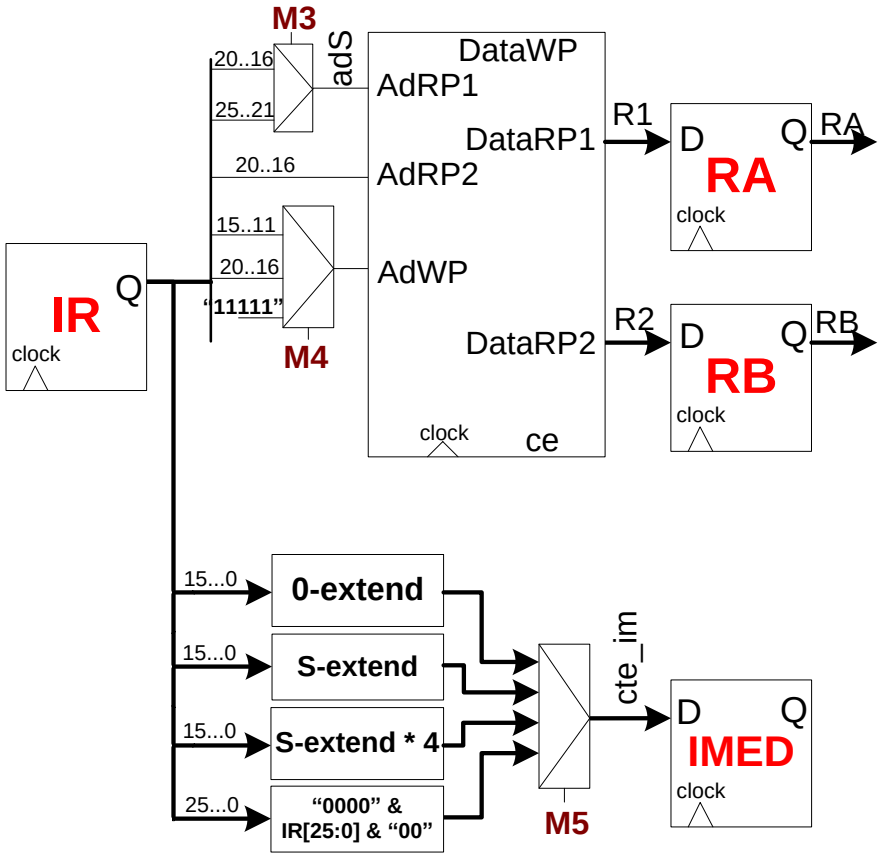


Porque M3 é necessário?

- Devido às instruções de deslocamento (sll, srl, etc.)
- RB e IMED compartilham uma entrada da ULA (ver Terceiro estágio)
- As instruções de deslocamento usam a constante para especificar a quantidade de bits a deslocar
- Logo, o registrador fonte, que está nos bits 20..16, deve ser lido e gravado em RA

Segundo estágio

- **Multiplexador M5**
  - **Determina o valor do dado imediato (constante)**

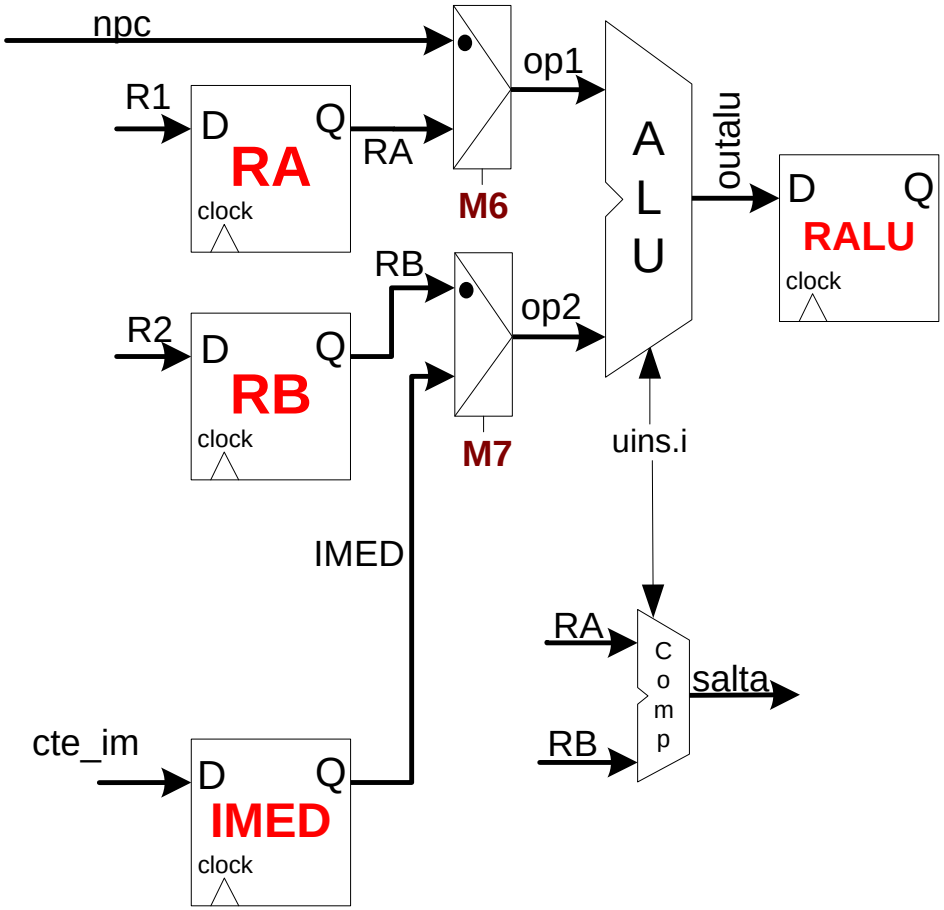


- **Pseudo-código VHDL**

```
cte_im <= sign-extend(29 downto 0)  & "00" when inst_branch='1'         else
          "0000" & IR(25 downto 0)  & "00" when uins.i=J or uins.i=JAL else
          x"0000" & IR(15 downto 0)        when uins.i={ANDI,ORI,XORI} else
          sign-extend;
```

Terceiro estágio

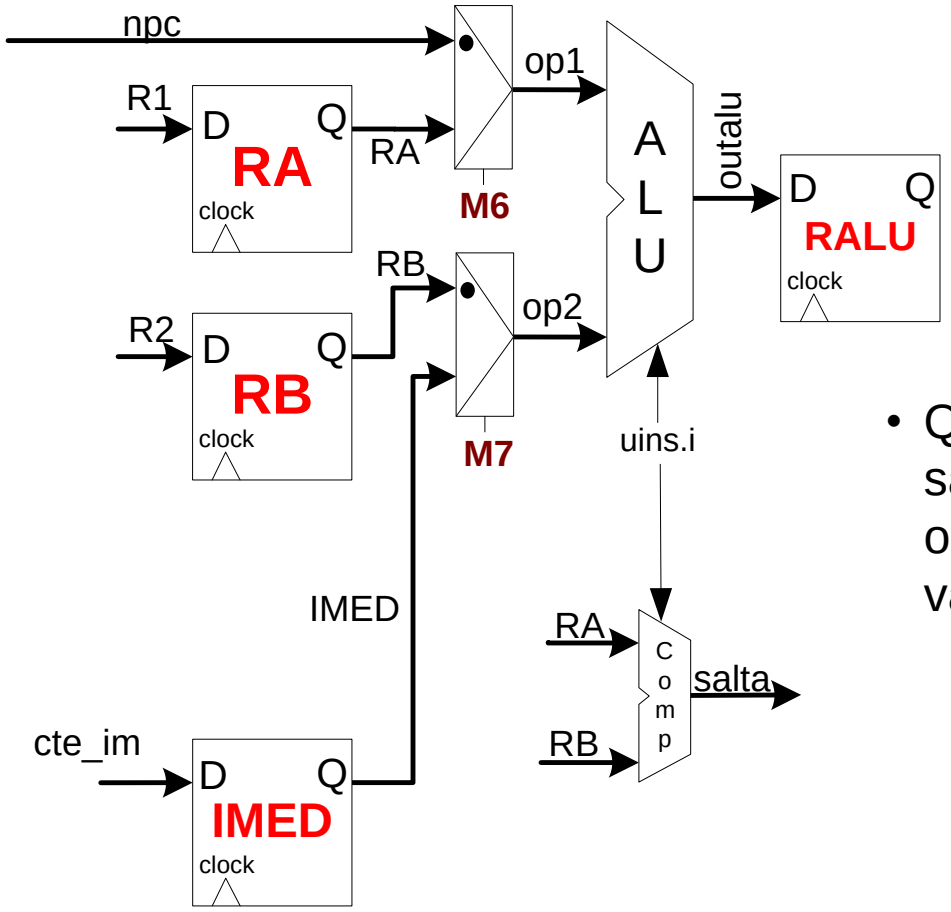
- **Comum a todas as instruções**
- **Realiza a operação com a ULA**
- **Determina um *flag* (salta) que indica se a condição de uma instrução de salto é verdadeira ou não**



Terceiro estágio

- **Controle do multiplexador M6**

```
op1 <= npc  when inst_branch='1' else  RA;
```

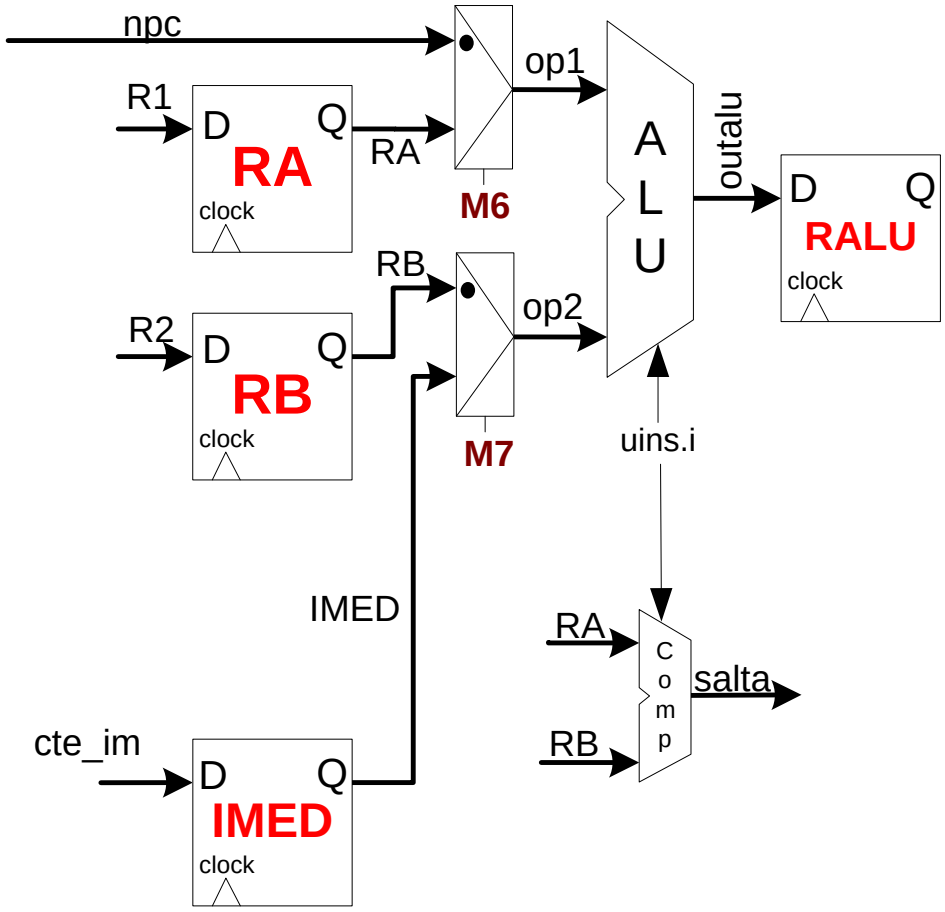


- Quando se tem uma instrução de salto condicional, a ULA determina o endereço do salto, somando o valor do PC à constante imediata

Terceiro estágio

- **Controle do multiplexador M7**

```
op2 <= RB when i={ADDU, SUBU, AND, OR, XOR, NOR, SLTU, SLT, JR, SLLV,
                                                SRAV, SRLV} else
       IMED;
```

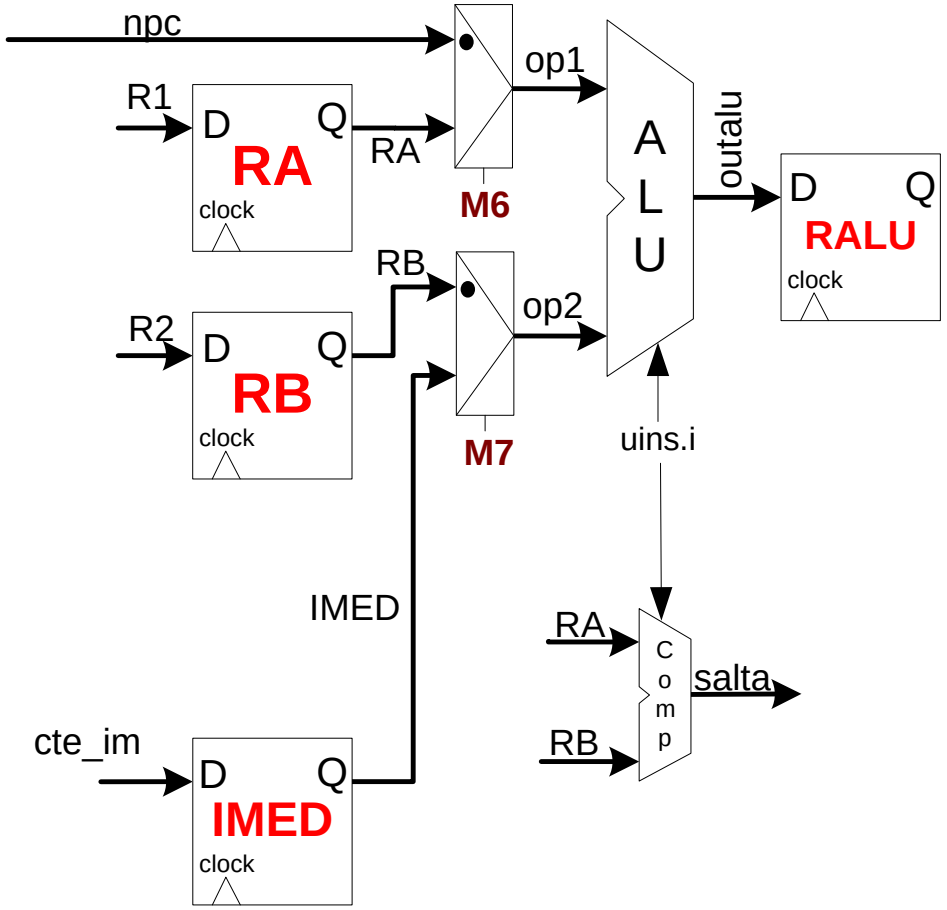


Terceiro estágio

- **Comparador**

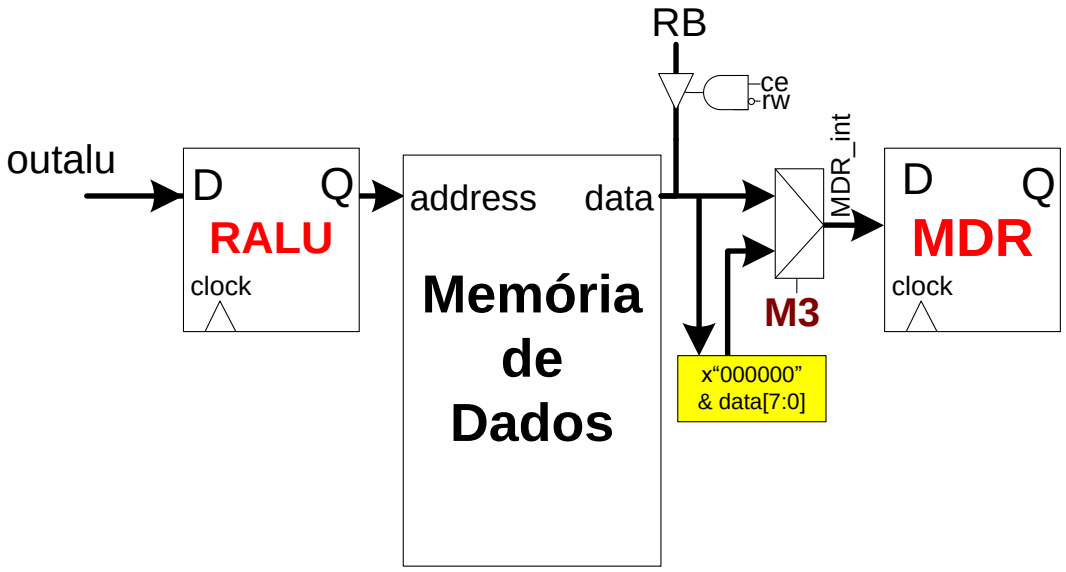
```
salta <=  '1' when ( (RA=RB  and uins.i=BEQ)  or (RA>=0  and uins.i=BGEZ) or
                     (RA<=0  and uins.i=BLEZ) or (RA/=RB and uins.i=BNE) ) else
          '0';
```

- O valor poderia ser armazenado em um flip-flop



Quarto estágio

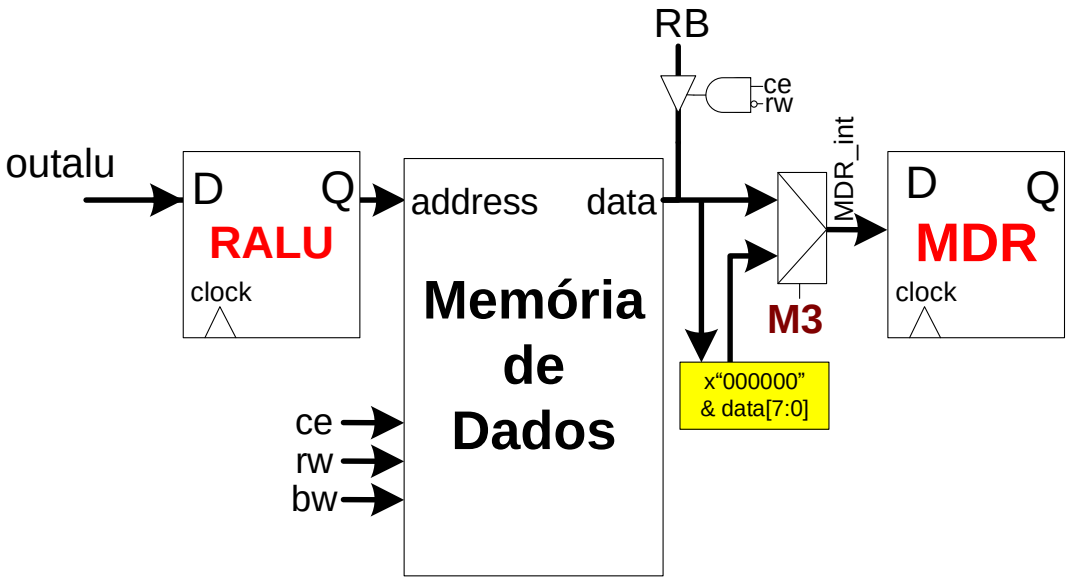
- **Utilizado apenas pelas instruções LBU/LW/SW/SB**
- **Leitura (LB/LW)**
  - **Valor endereçado pelo registrador RALU é lido da memória de dados e escrito no registrador MDR**
  - **Valor endereçado pelo registrador RALU recebe o conteúdo do registrador RB quando for instrução de escrita**



Quarto estágio

- **Acesso a byte**

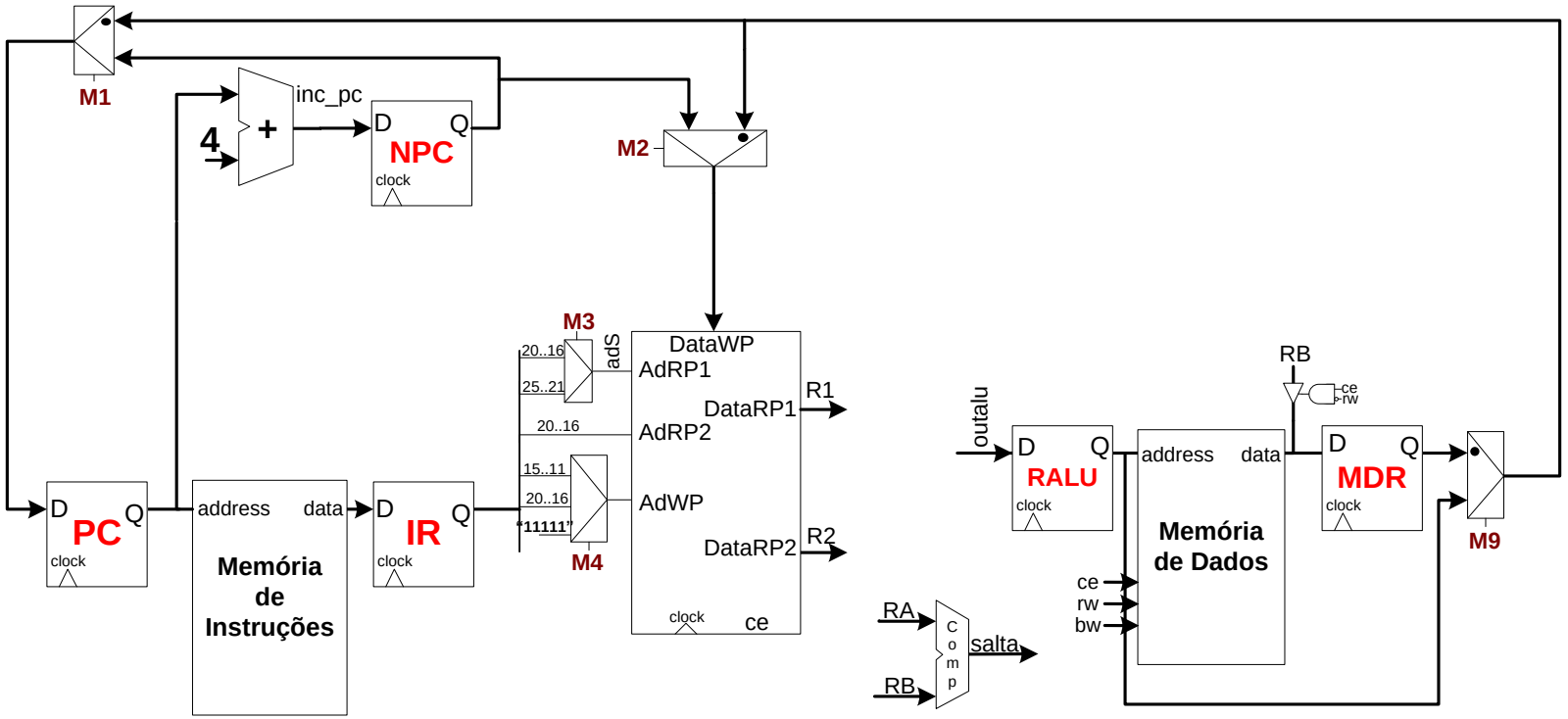
  - **Load Byte Unsigned: na instrução LBU, grava-se em MDR o byte menos significativo (LSB) e coloca-se em zero os 3 bytes mais significativos (MSBs com x “000000”)**



  - **Store Byte (instrução SB): a memória recebe 32 bits e um sinal de controle (bw=0) para seleciona escrever apenas byte menos significativo**

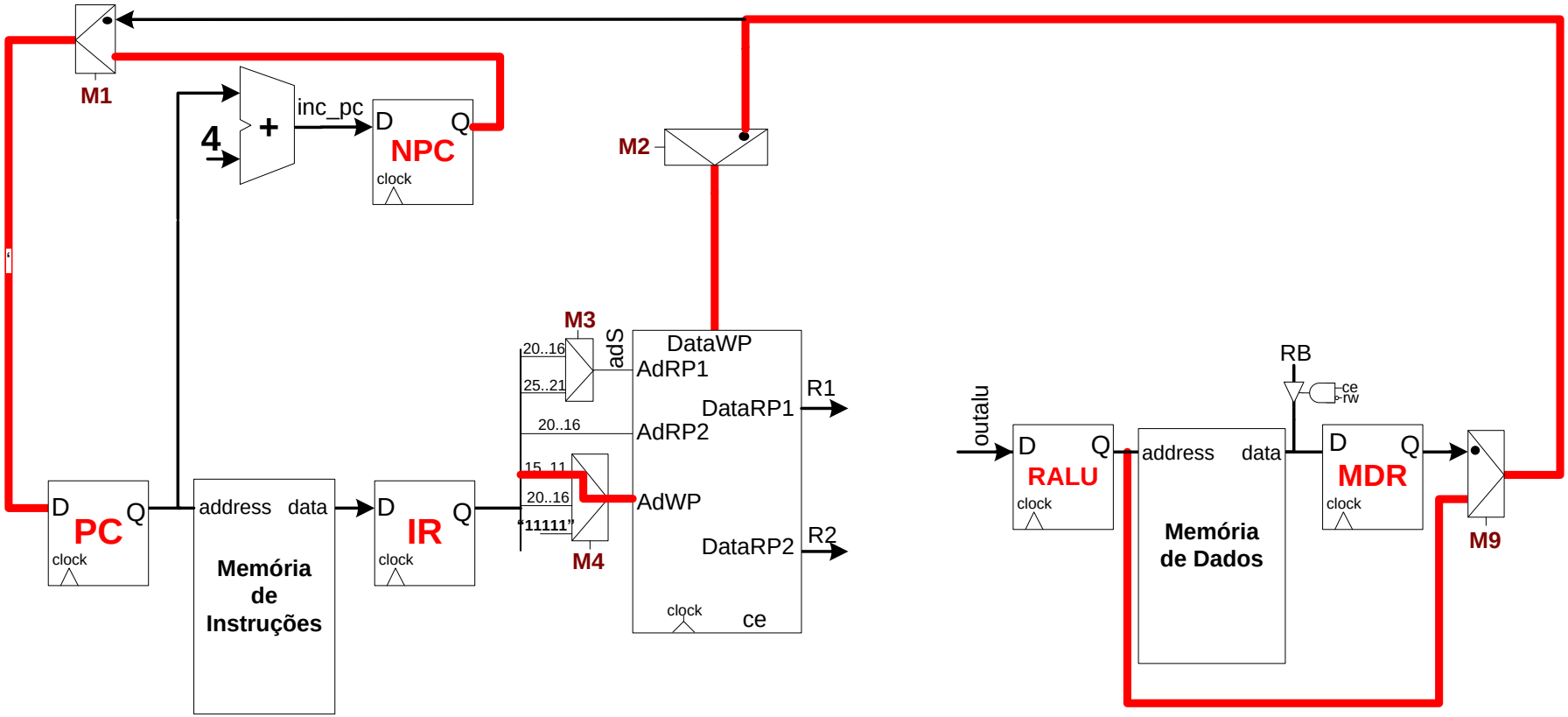
Quinto estágio

- **Estágio que conclui a execução de uma determinada instrução**
- **Denomina-se em inglês *write-back***
- **Depende da instrução corrente**



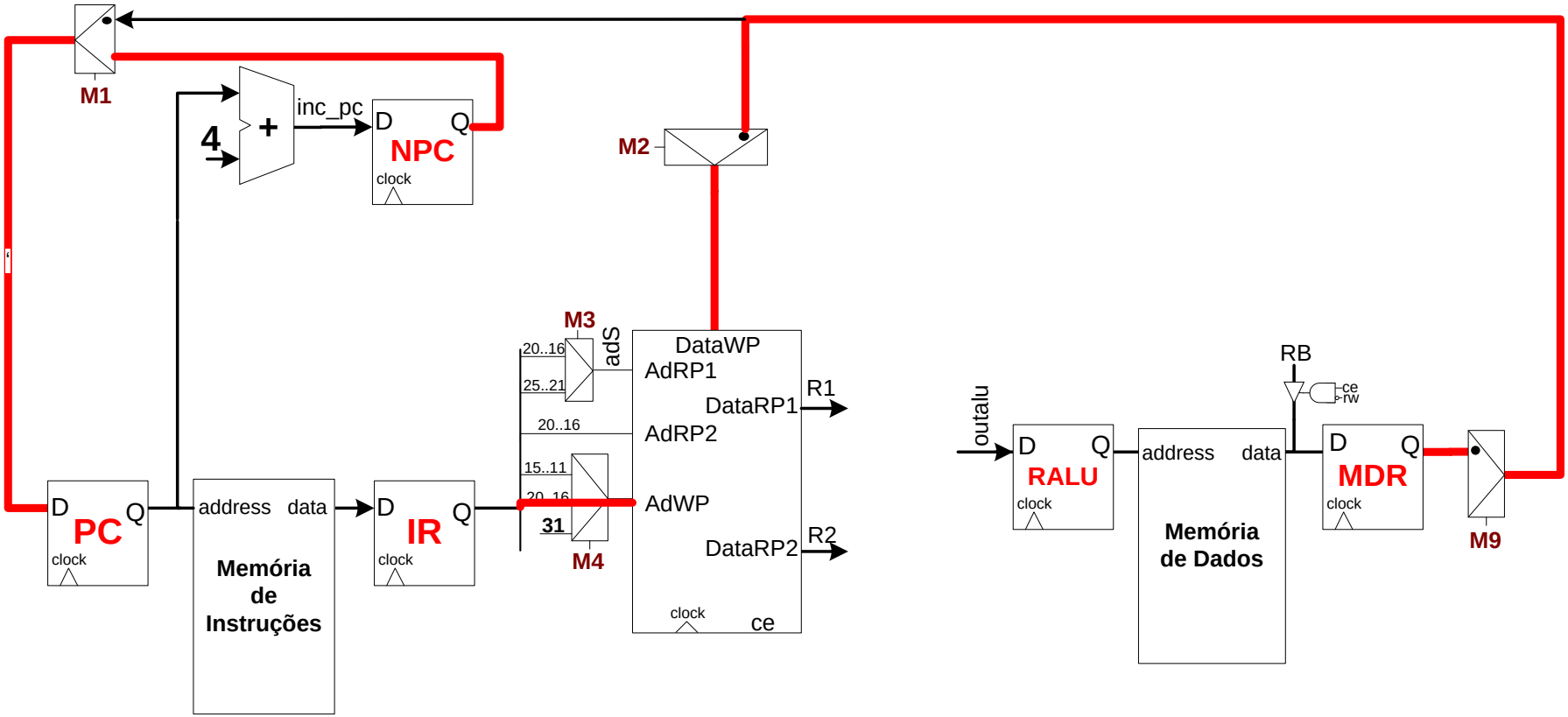
Quinto estágio

- **Instruções lógico-aritméticas**
  - **O resultado da operação da ALU é escrito no banco de registradores**
  - **O valor do NPC (PC incrementado de 4) é escrito no PC**



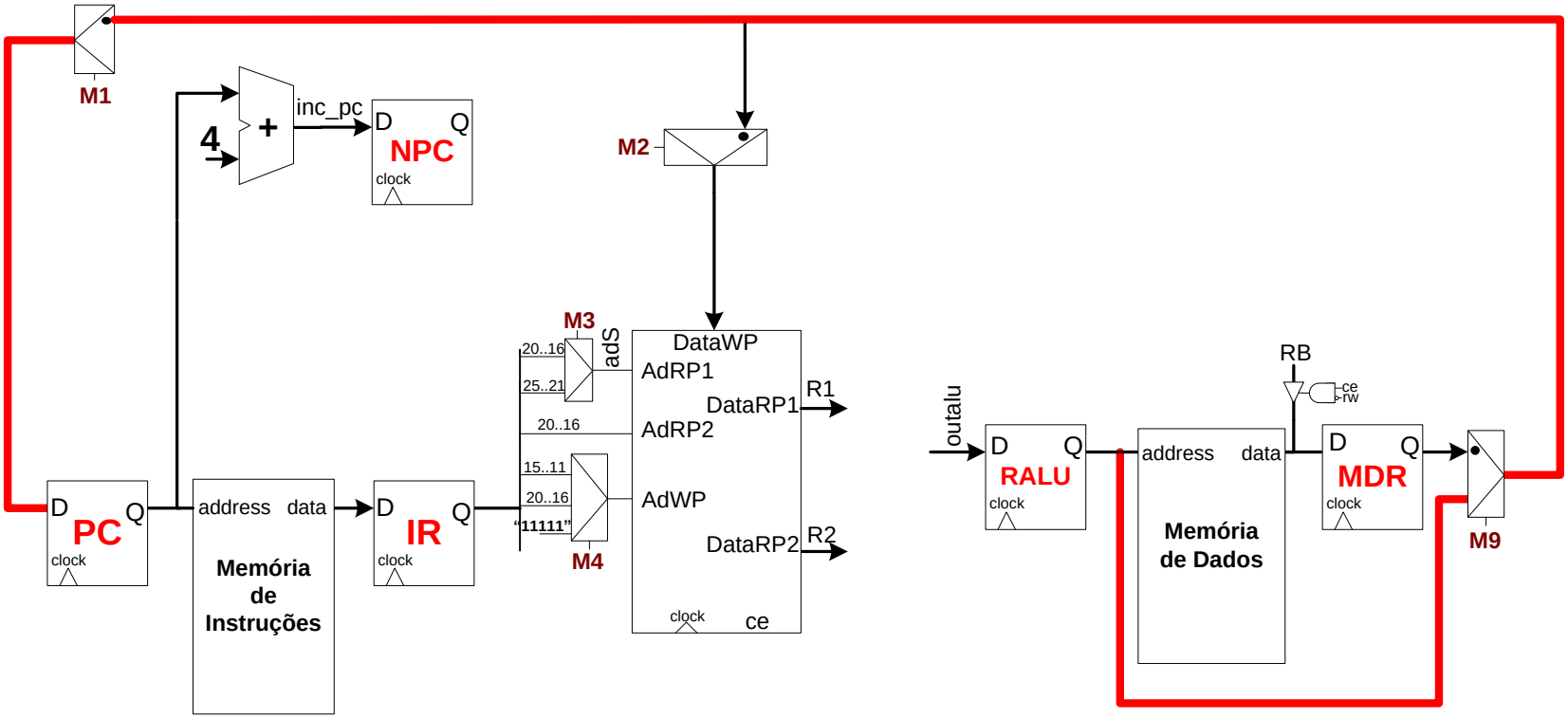
Quinto estágio

- **Instruções LW/LB**
  - **Valor armazenado em MDR é escrito no banco de registradores**
  - **O valor do NPC (PC incrementado de 4) é escrito no PC**



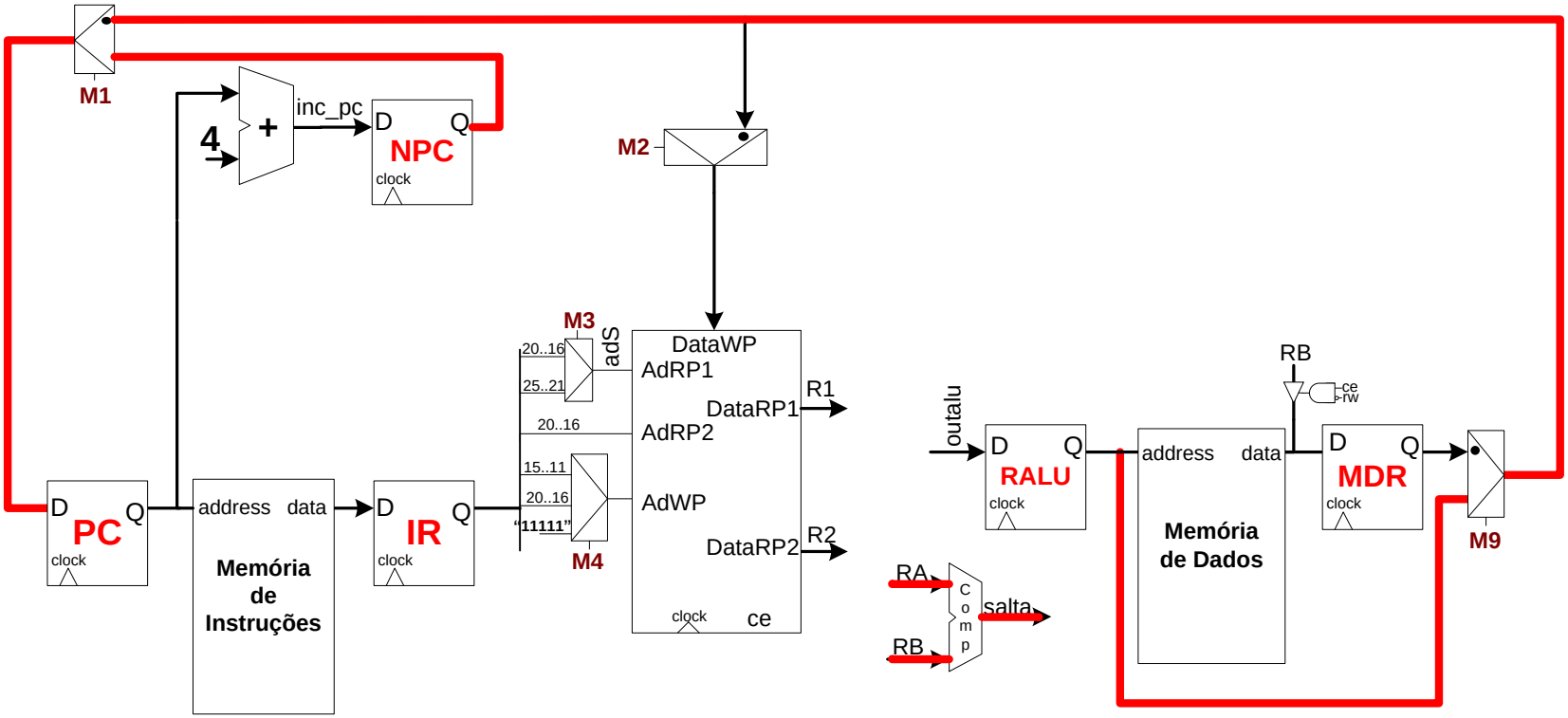
Quinto estágio

- **Instruções de salto incondicional**
  - **J / JR**
  - **PC recebe valor contido no registrador Ralu**



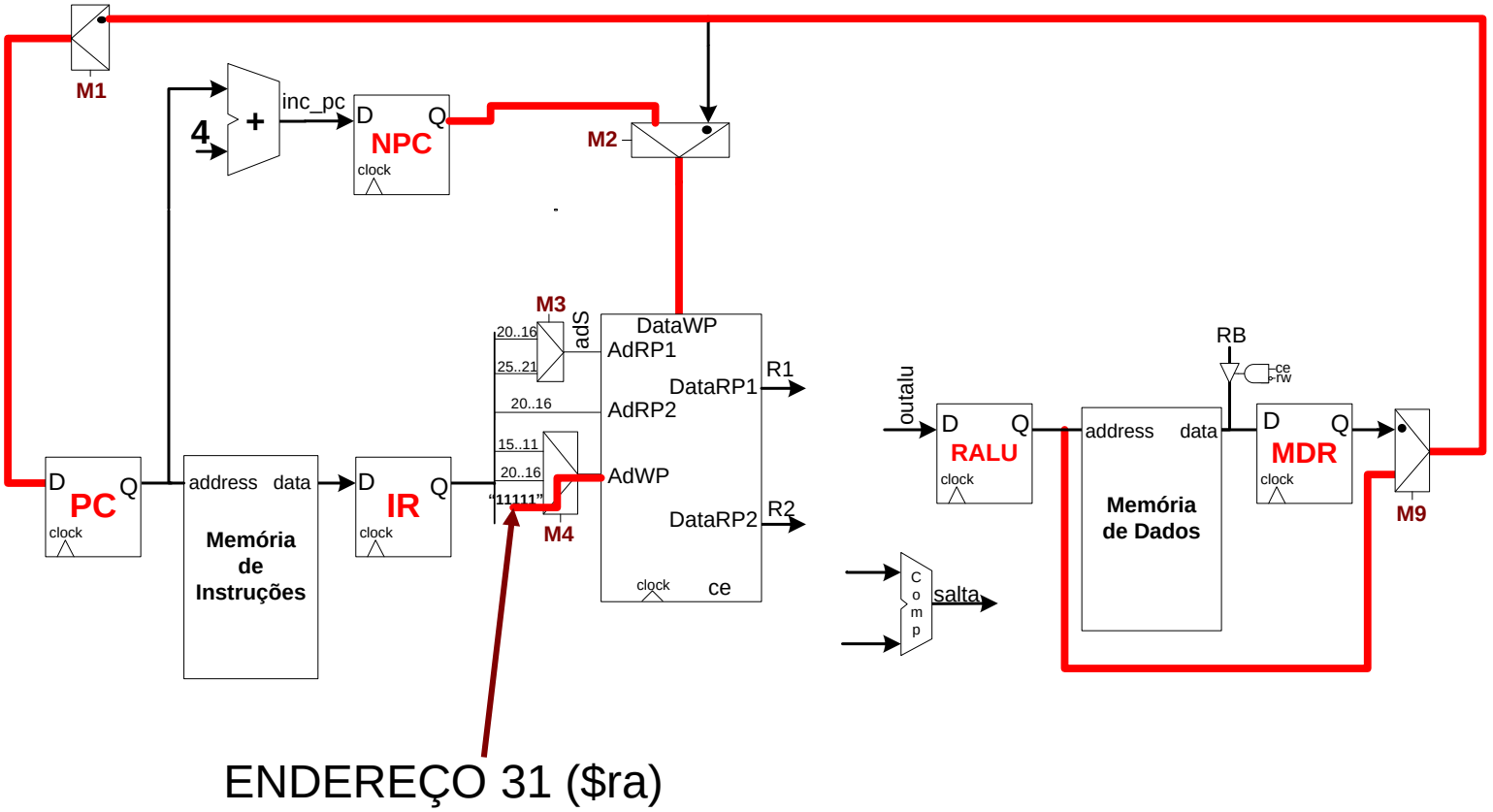
Quinto estágio

- **Instruções de salto condicional**
  - **O PC pode receber o NPC ou o endereço destino, calculado na ULA**
  - **O controle do mux M1 é a saída do comparador**



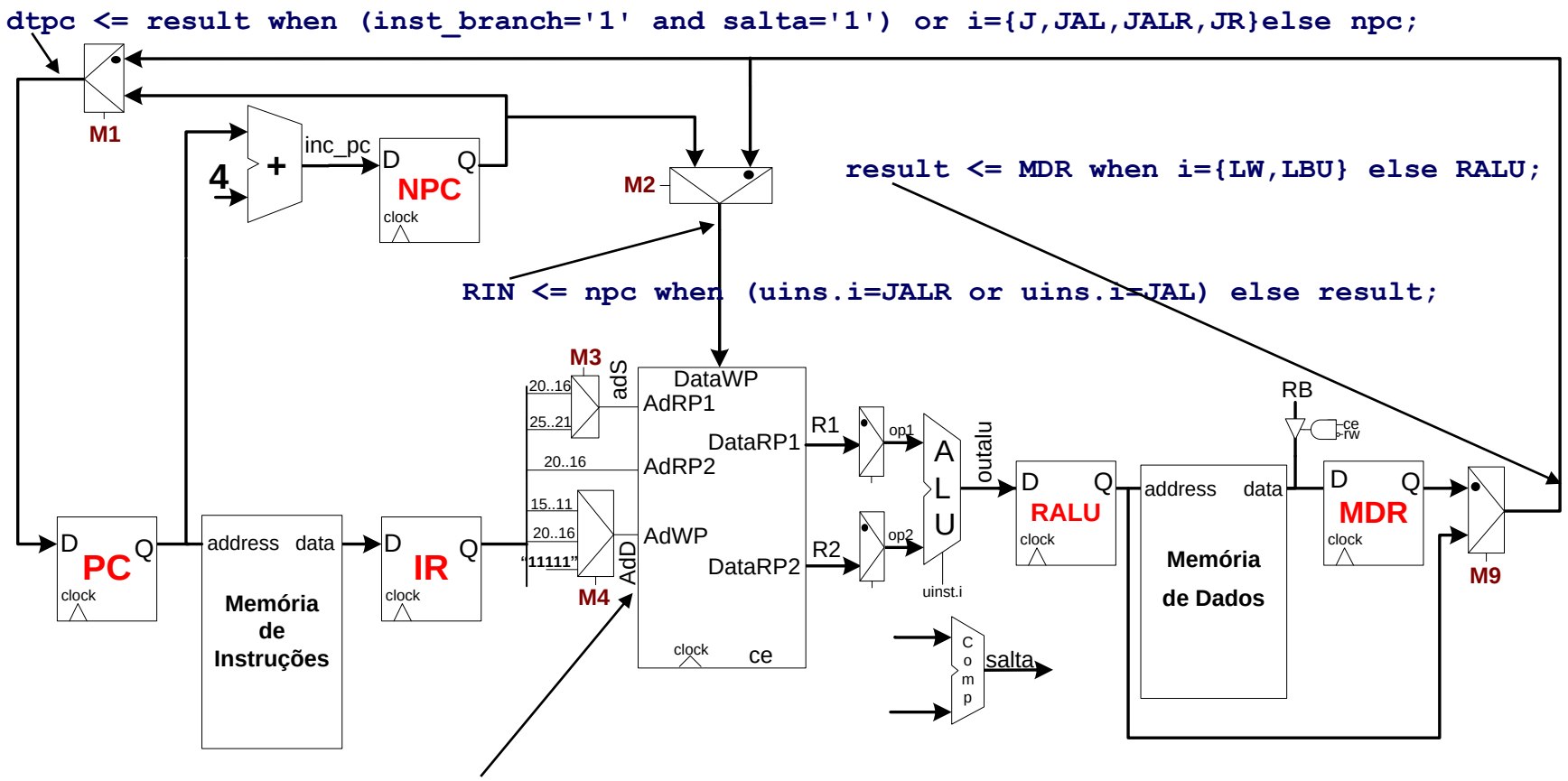
Quinto estágio

- **Instruções JAL (Jump and Link)**
  - **O PC recebe o endereço da rotina**
  - **O registrador $ra (endereço “11111” recebe o NPC (PC+4, o endereço de retorno da subrotina)**

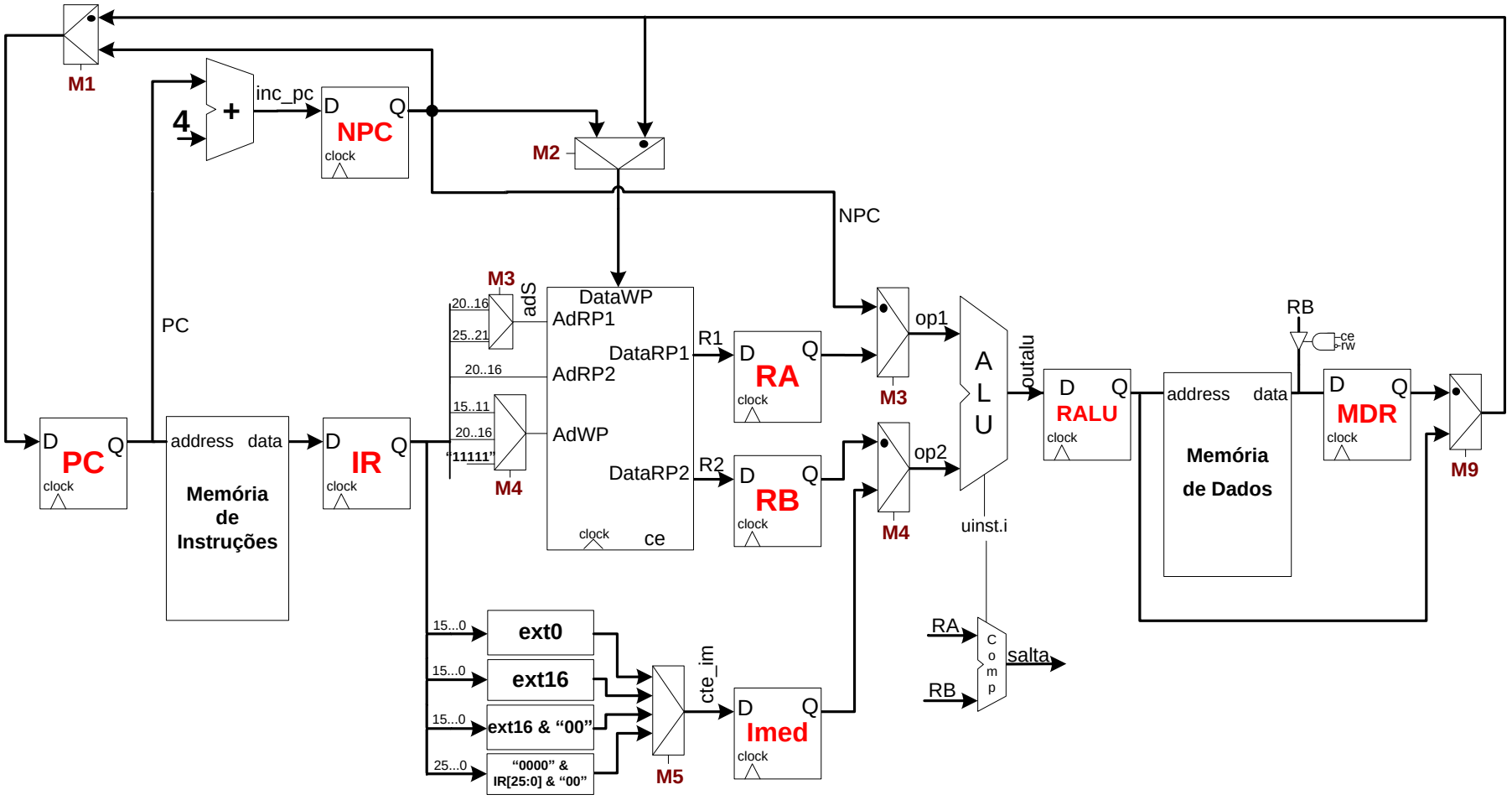


Quinto estágio

- **Multiplexadores de quinto estágio**



Bloco de dados - quase completo

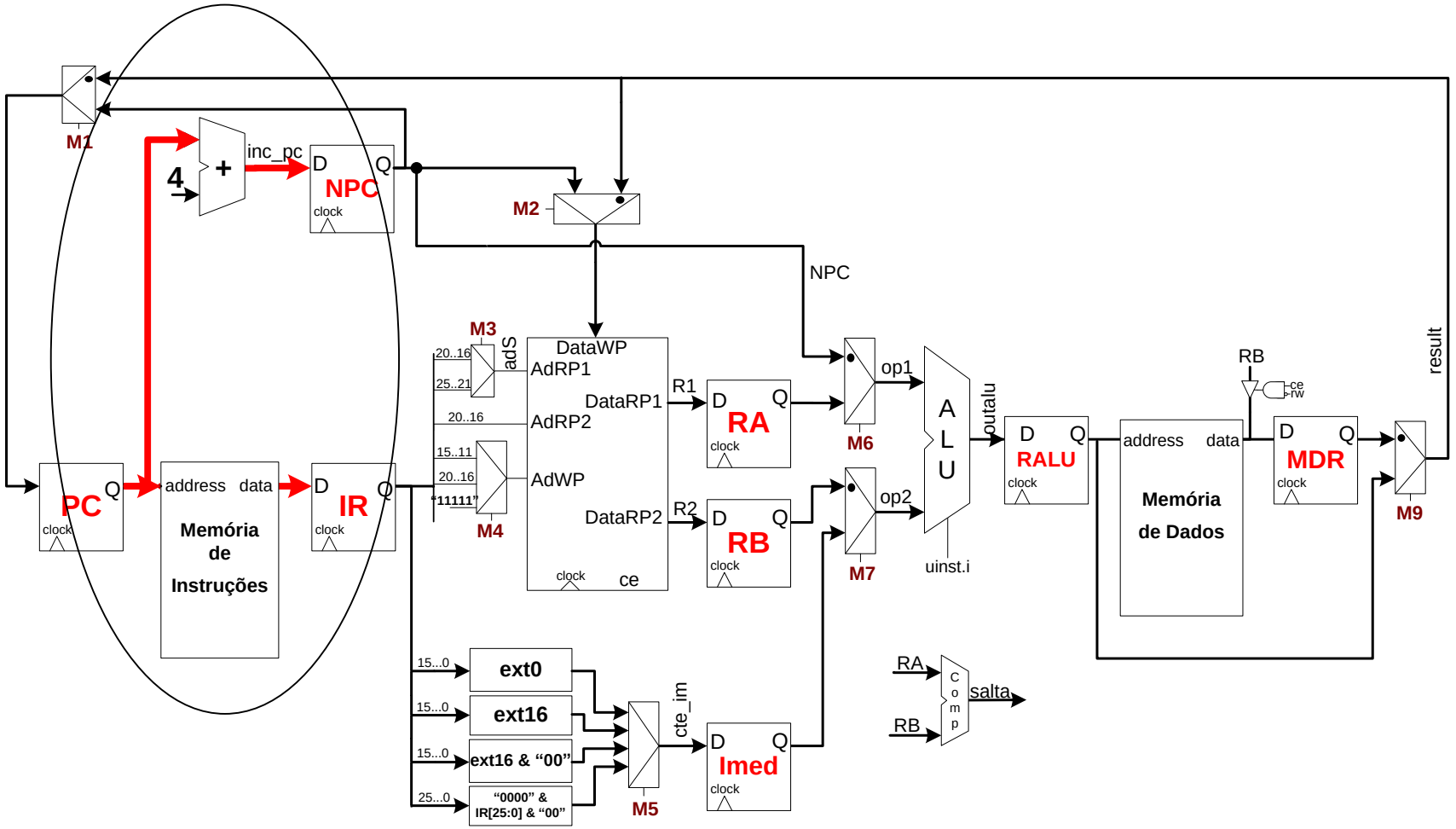


Principais Diferenças MIPS: monociclo X multiciclo

- **Registradores temporários para isolar estágios**
  - **IR, NPC, RA, RB, IMED, RALU, MDR**
- **Bloco de controle (BC) – como sinais de controle devem ser gerados apenas em ciclos específicos de cada instrução, o BC deixa de ser combinacional e passar a ser uma máquina de estados finita**
- **Atualização do PC adiada para o fim da instrução, porquê?**
- **Novos multiplexadores – sem relação com mono-multiciclo, consequência da versão multiciclo ser mais completa no suporte à arquitetura MIPS**

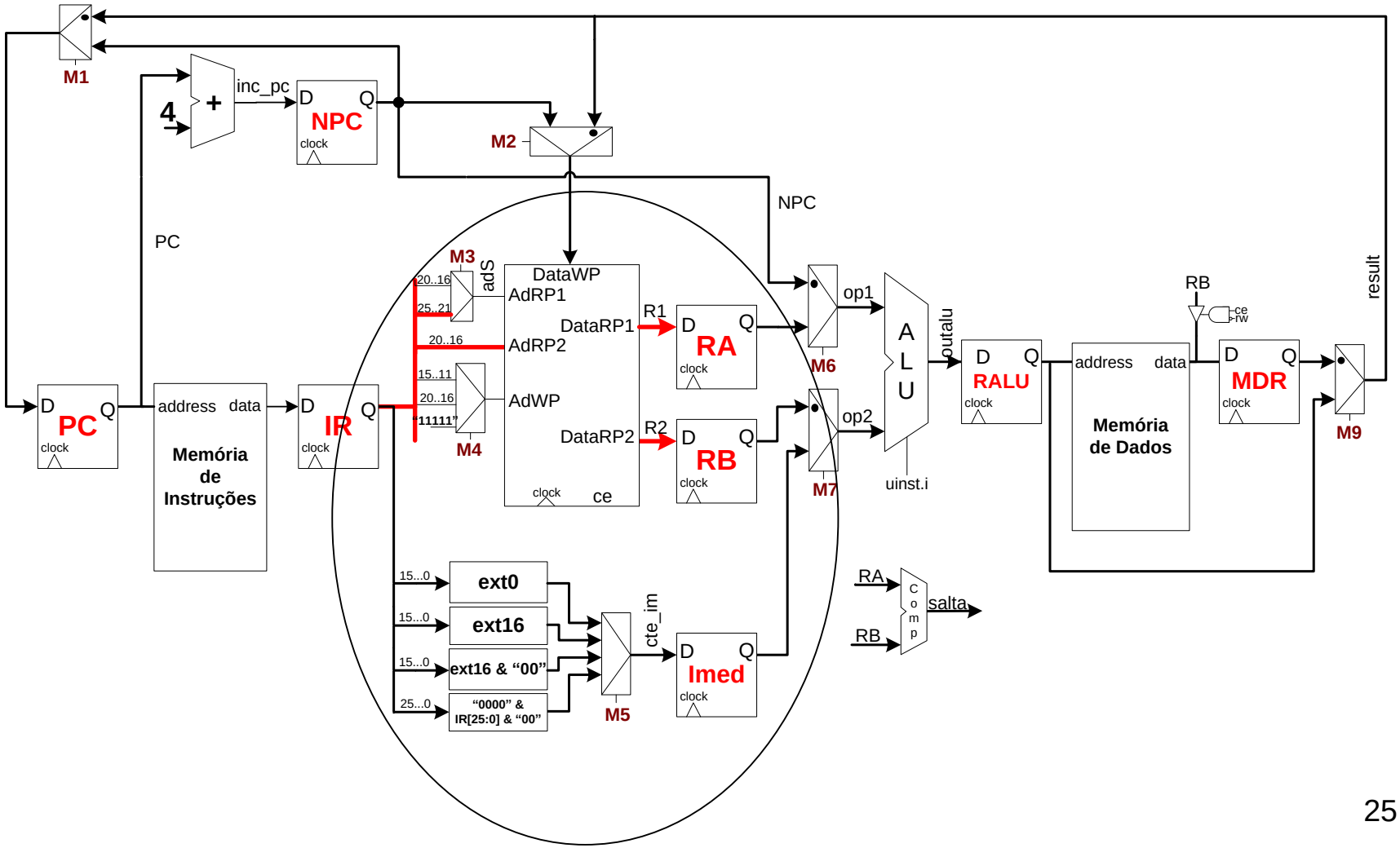
Execução ciclo de clock a ciclo de clock (lógico-aritméticas-R)

- **Primeiro ciclo: busca da operação *(fetch)***



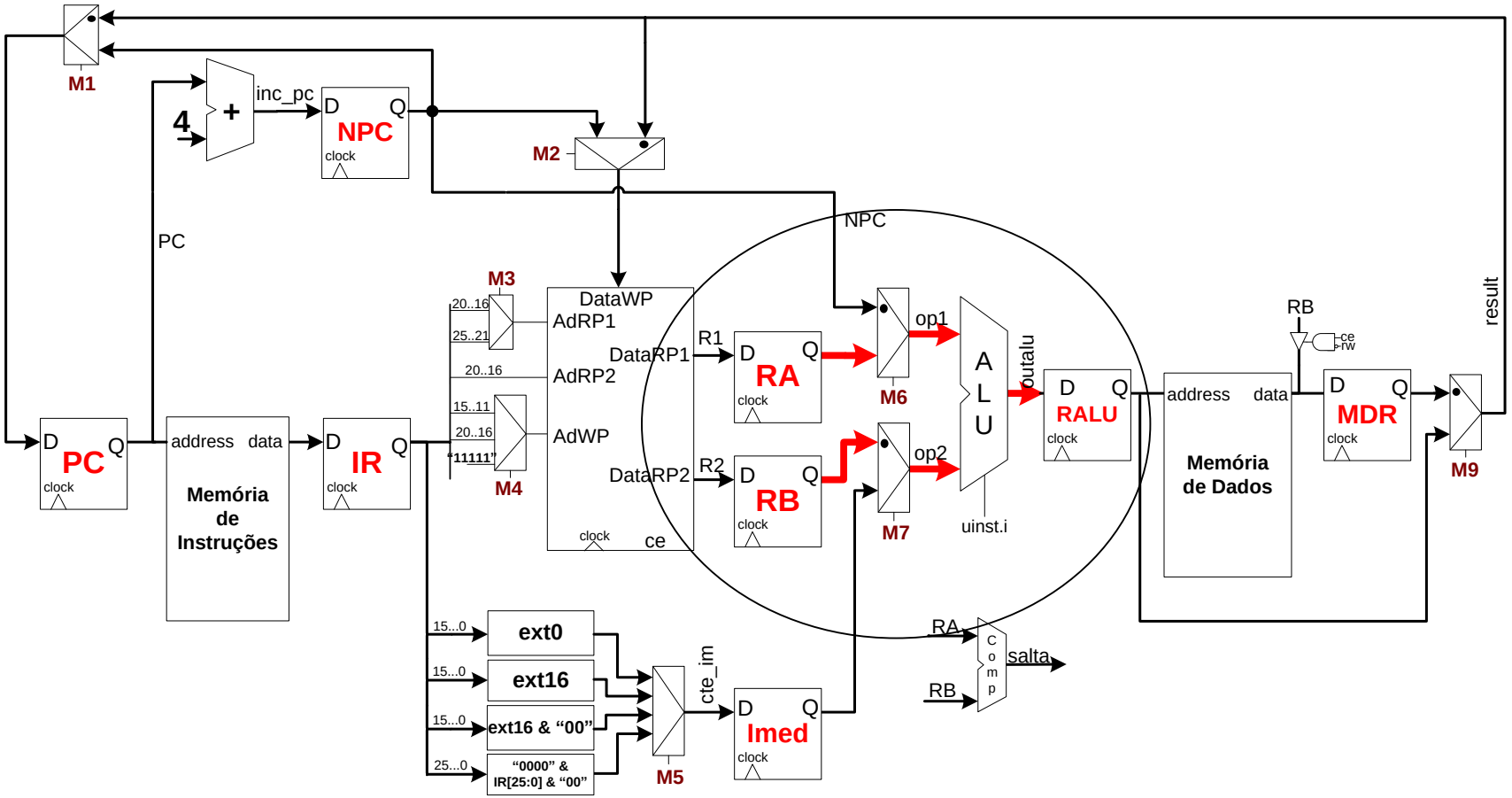
Execução ciclo de clock a ciclo de clock (lógico-aritméticas-R)

- **Leitura dos registradores**



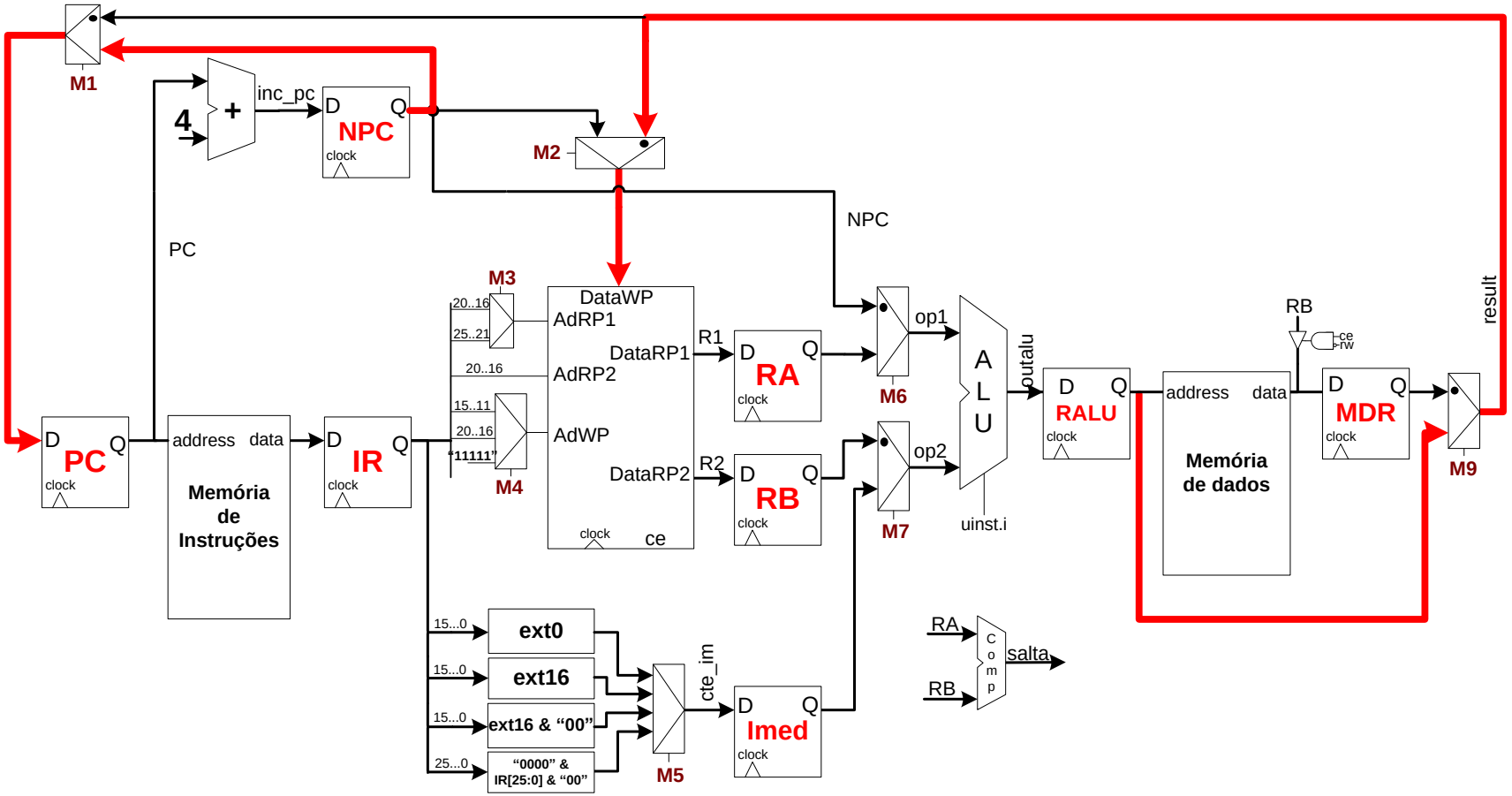
Execução ciclo de clock a ciclo de clock (lógico-aritméticas-R)

- **Terceiro ciclo: operação com a ULA**



Execução ciclo de clock a ciclo de clock (lógico-aritméticas-R)

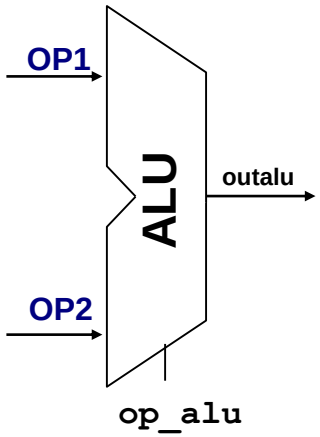
- **Quarto ciclo: atualiza PC e atualiza o banco de registradores**



ULA

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use work.p_MR2.all;

entity alu is
       port( op1, op2 : in reg32;
             outalu :   out reg32;
             op_alu : in inst_type
           );
end alu;

architecture alu of alu is
   signal menorU, menorS : std_logic ;
begin

    menorU <=  '1' when op1 < op2 else '0';

    menorS <=  '1' when ieee.std_Logic_signed."<"(op1,  op2) else '0' ; -- signed
```

Atenção: operações sem signal



COMPARAÇÃO sem sinal

COMPARAÇÃO com sinal – redefinição do operador <

- **Observar uso do tipo *inst_type* na entity**

ULA

- **Obervar:**
  - **uso de concatenação nos SLT*xx***
  - **operações de deslocamento**

```
outalu <=  op1 - op2                    when  op_alu=SUBU                    else
           op1 and op2                  when  op_alu=AAND  or op_alu=ANDI    else
           op1 or  op2                  when  op_alu=OOR   or op_alu=ORI     else
           op1 xor op2                  when  op_alu=XXOR  or op_alu=XORI    else
           op2(15 downto 0) & x"0000"   when  op_alu=LUI                     else
           (0=>menorU, others=>'0')     when  op_alu=SLTU  or op_alu=SLTIU   else
           (0=>menorS, others=>'0')     when  op_alu=SLT   or op_alu=SLTI    else
           op1(31 downto 28) & op2(27 downto 0)     when  op_alu=J           else
           op1                          when  op_alu=JR    or op_alu=JALR    else
           to_StdLogicVector( to_bitvector(op1) sll CONV_INTEGER(op2(10 downto 6)))
                                                    when  op_alu=SSLL    else
           to_StdLogicVector( to_bitvector(op1) srl CONV_INTEGER(op2(10 downto 6)))
                                                    when  op_alu=SSRL    else
           op1 + op2;                -- default
end alu;
```

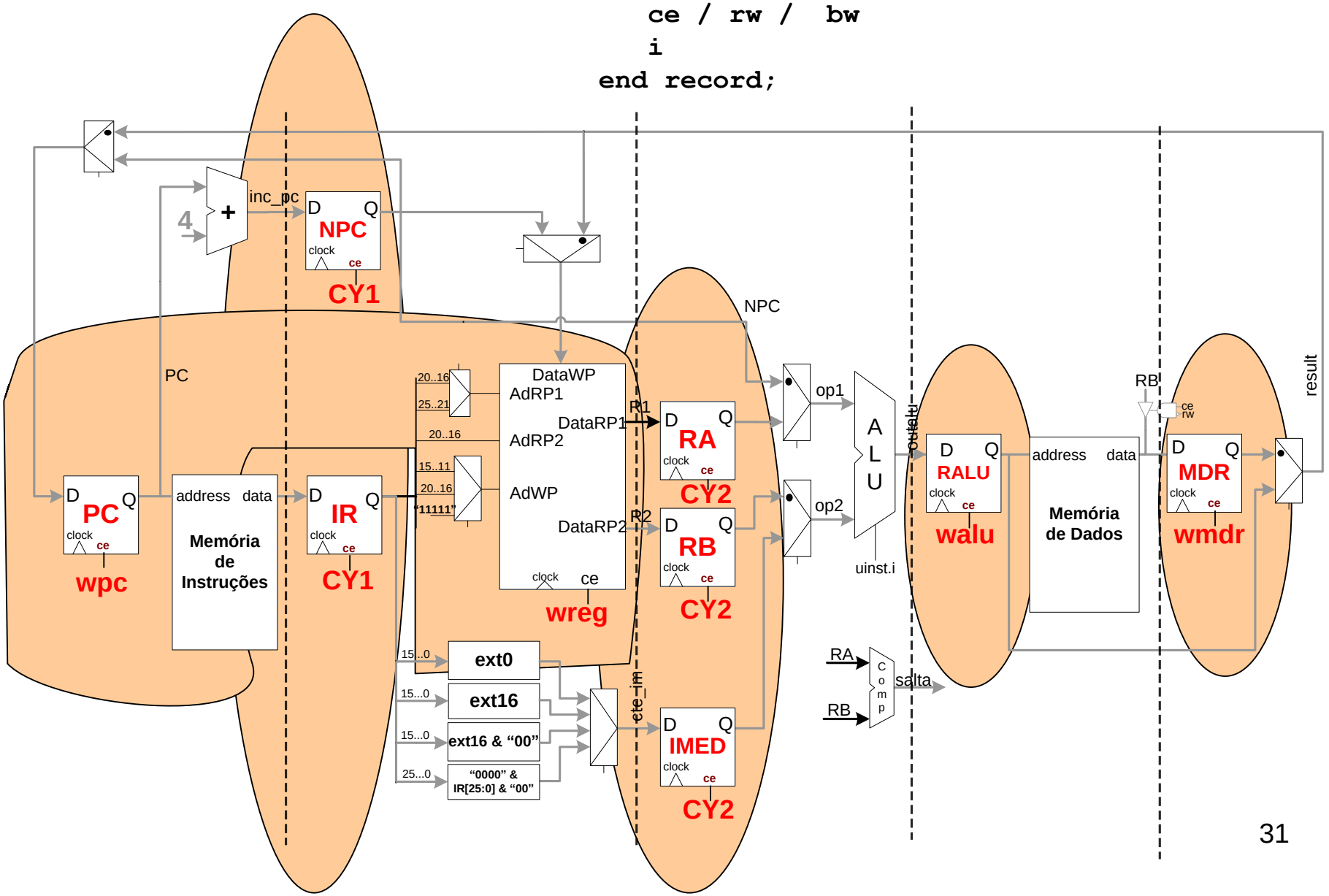
Bloco de controle do Processador Multi-Ciclo

- Microinstrução definida no package

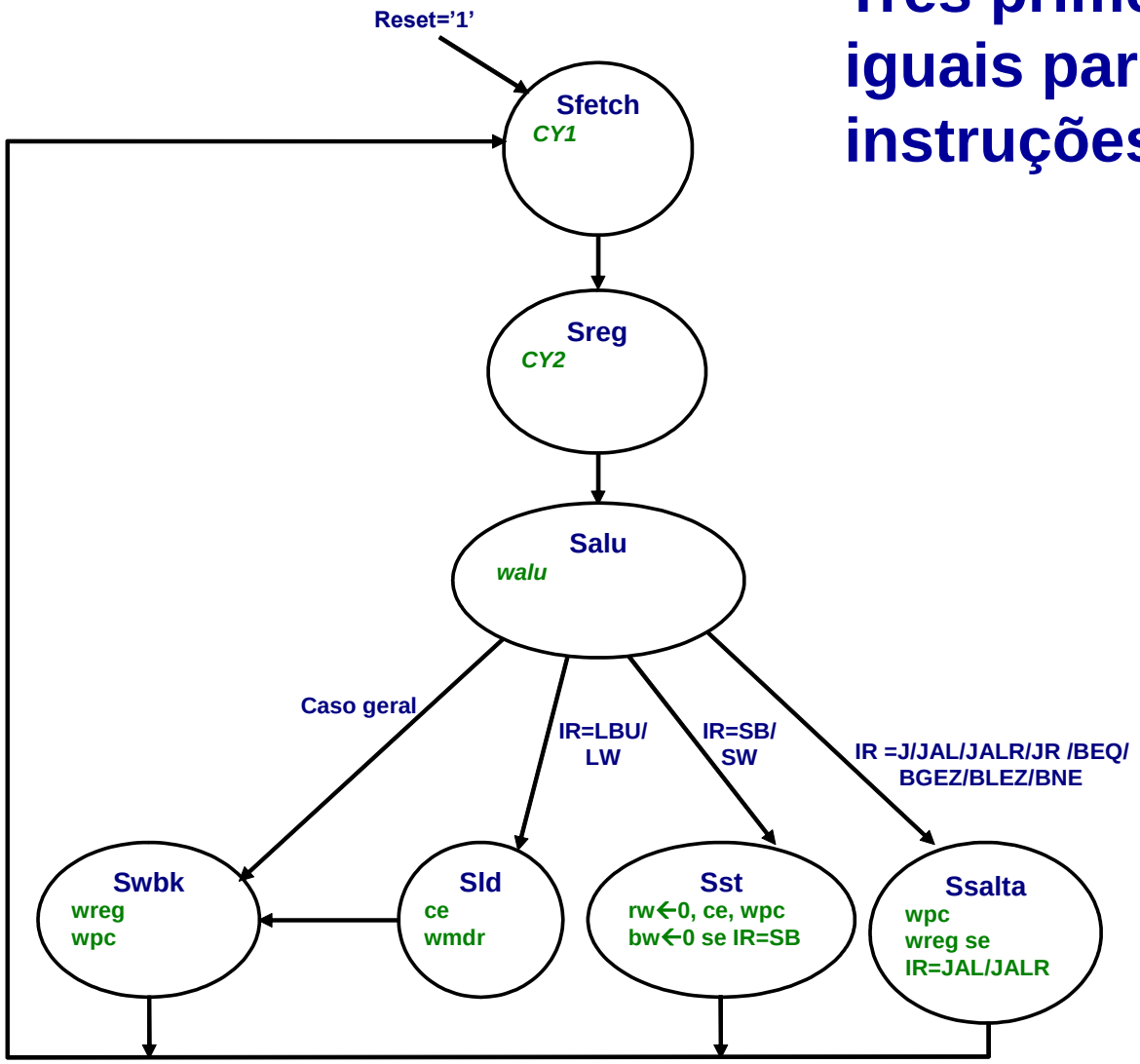
```
type microinstruction is record
   CY1:    std_logic;       -- write enable for regs of 1st stage
   CY2:    std_logic;       --    "  enable for regs of 2nd stage
   walu:   std_logic;       --    "  enable for third stage reg
   wmdr:   std_logic;       --    "  enable for fourth stage reg
   wpc:    std_logic;       -- PC write enable
   wreg:   std_logic;       -- Register Bank write enable
   ce:     std_logic;       -- Data Memory CE and R/W controls
   rw:     std_logic;
   bw:     std_logic;       -- Data Memory Byte/Word write control
   i:      inst_type;       -- instruction identification
end record;
```

Sinais de controle

```
type microinstruction is record
   CY1  / CY2/  wula / wmdr / wpc / wreg
   ce / rw /  bw
   i
end record;
```



Bloco de controle do Processador Multi-Ciclo



- **Três primeiros ciclos iguais para todas as instruções**

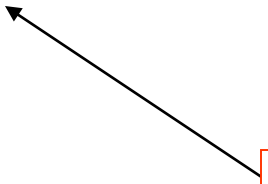
Entidade/Arquitetura do bloco de controle

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use work.p_MR2.all;

entity control_unit is
        port(   ck, rst : in std_logic;
                uins : out microinstruction;
                ir : in reg32
            );
end control_unit;

architecture control_unit of control_unit is

   type type_state is (Sfetch, Sreg, Salu, Swbk, Sld, Sst, Ssalta);

   signal PS, NS : type_state;
   signal i : inst_type;

begin
```

ENUMERAÇÃO PARA A MÁQUINA DE ESTADOS

Primeira parte – decodificação de instruções

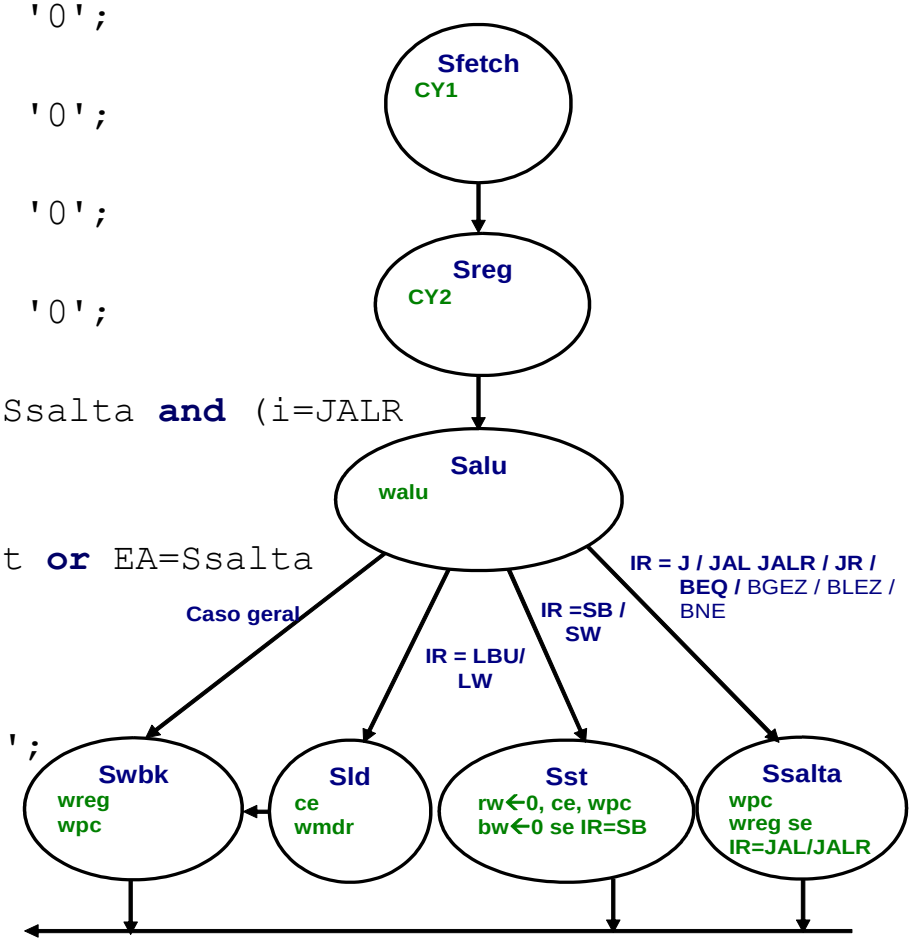
```
i <=    ADDU   when ir(31 downto 26)="000000" and ir(5 downto 0)="100001" else
        SUBU   when ir(31 downto 26)="000000" and ir(5 downto 0)="100011" else
        AAND   when ir(31 downto 26)="000000" and ir(5 downto 0)="100100" else
        OOR    when ir(31 downto 26)="000000" and ir(5 downto 0)="100101" else
        XXOR   when ir(31 downto 26)="000000" and ir(5 downto 0)="100110" else
        SSLL   when ir(31 downto 21)="00000000000" and ir(5 downto 0)="000000" else
        SSRL   when ir(31 downto 21)="00000000000" and ir(5 downto 0)="000010" else
        ADDIU  when ir(31 downto 26)="001001" else
        ANDI   when ir(31 downto 26)="001100" else
        ....
        JR     when ir(31 downto 26)="000000" and ir(5 downto 0)="001000" else
        invalid_instruction ;

assert i /= invalid_instruction
      report "******************* INVALID INSTRUCTION *************"
      severity error;

uins.i <= i;
```



COMPONENTE PRINCIPAL DA MICROINSTRUÇÃO

Segunda parte – escrita nos registradores / acesso memória

```
uins.CY1   <= '1' when EA=Sfetch else   '0';

uins.CY2   <= '1' when EA=Sreg   else   '0';

uins.walu  <= '1' when EA=Salu else     '0';

uins.wmdr  <= '1' when EA=Sld  else     '0';

uins.wreg   <= '1' when EA=Swbk or (EA=Ssalta and (i=JALR
                 or i=JAL)) else '0';

uins.wpc   <= '1' when EA=Swbk or EA=Sst or EA=Ssalta
               else '0';


uins.rw    <= '0' when EA=Sst  else  '1';

uins.ce    <= '1' when EA=Sld or EA=Sst
               else '0';

uins.bw    <= '0' when (EA=Sst and i=SB)
               else '1';
```



Exercício: Máquina de Estados

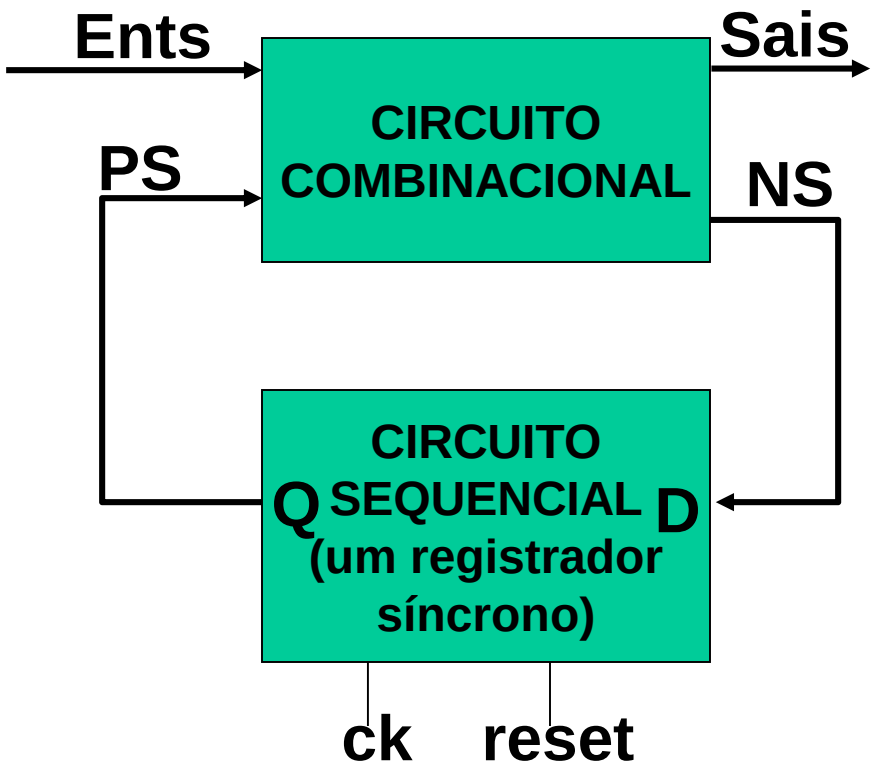
```
entity fsm is port(X, rst, ck : in std_logic;
                   Z: out std_logic);
end;

architecture A of fsm is
    type STATES is (S0, S1, S2, S3);
    signal scurrent, snext : STATES;
begin
      process(rst, ck)
         begin
            if rst='1'                  then   scurrent <= S0;
            elsif ck'event and ck='1'   then   scurrent <= snext;
            end if;
       end process;

      process(scurrent, X)
       begin
         case scurrent is
           when S0 =>  if X='0' then Z<='0'; snext <= S1;
                               else Z<='1'; snext <= S2;
                       end if;
           when S1 =>  if X='0' then Z<='1'; snext <= S2;
                               else Z<='0'; snext <= S1;
                       end if;
           when S2 =>  if X='0' then Z<='1'; snext <= S2;
                               else Z<='0'; snext <= S3;
                       end if;
           when S3 =>  if X='0' then Z<='0'; snext <= S0;
                               else Z<='0'; snext <= S1;
                       end if;
         end case;
       end process;
 end A;
```

1.Desenhe o diagrama de transição de estados da máquina de estados finitos ao lado.
2.Trata-se de uma máquina de Moore ou Mealy? Justifique.

Terceira parte – máquina de estados (1/2)

- Parte sequencial

```
process(rst, ck)
   begin
      if rst='1' then
        PS <= Sfetch;
      elsif ck'event and ck='1' then
        PS <= NS;
      end if;
   end process;
```



Terceira parte – máquina de estados (2/2)

- Parte combinacional (não considera MULTU e SUBU)

```
process(PS, i)
  begin
   case PS is
   when Sfetch => NS <= Sreg;
   when Sreg => NS <= Salu;
   when Salu =>  if (i=LW or i=LBU) then NS <= Sld;
                 elsif (i=SW or i=SB) then NS <= Sst;
                 elsif (i=J or i=JALR or i=JAL or i=JR or i=BEQ or i=BGEZ
                         or i=BLEZ or i=BNE) then NS <= Ssalta;
                 else                     PE <= Swbk;
                 end if;
   when Sld => NS <= Swbk;
   when Sst | Ssalta | Swbk => NS <= Sfetch;
    end case;
 end process;
end control_unit;
```

Tempo de execução de programas

- **Calcule o tempo de execução para o programa abaixo, supondo clock de 50 MHZ para a organização MIPS vista**

```
main:   la      $t0,array
        la      $t1,size
        lw      $t1,0($t1)
        la      $t2,const
        lw      $t2,0($t2)
loop:   blez    $t1,end
        lw      $t3,0($t0)
        addu    $t3,$t3,$t2
        sw      $t3,0($t0)
        addiu   $t0,$t0,4
        addiu   $t1,$t1,-1
        j       loop

end:    j       $ra

        .data
array:  .word   0x12 0xff 0x3 0x14 0x878 0x31  0x62 0x10 0x5 0x16 0x20
size:   .word   11
const:  .word   0x100
```

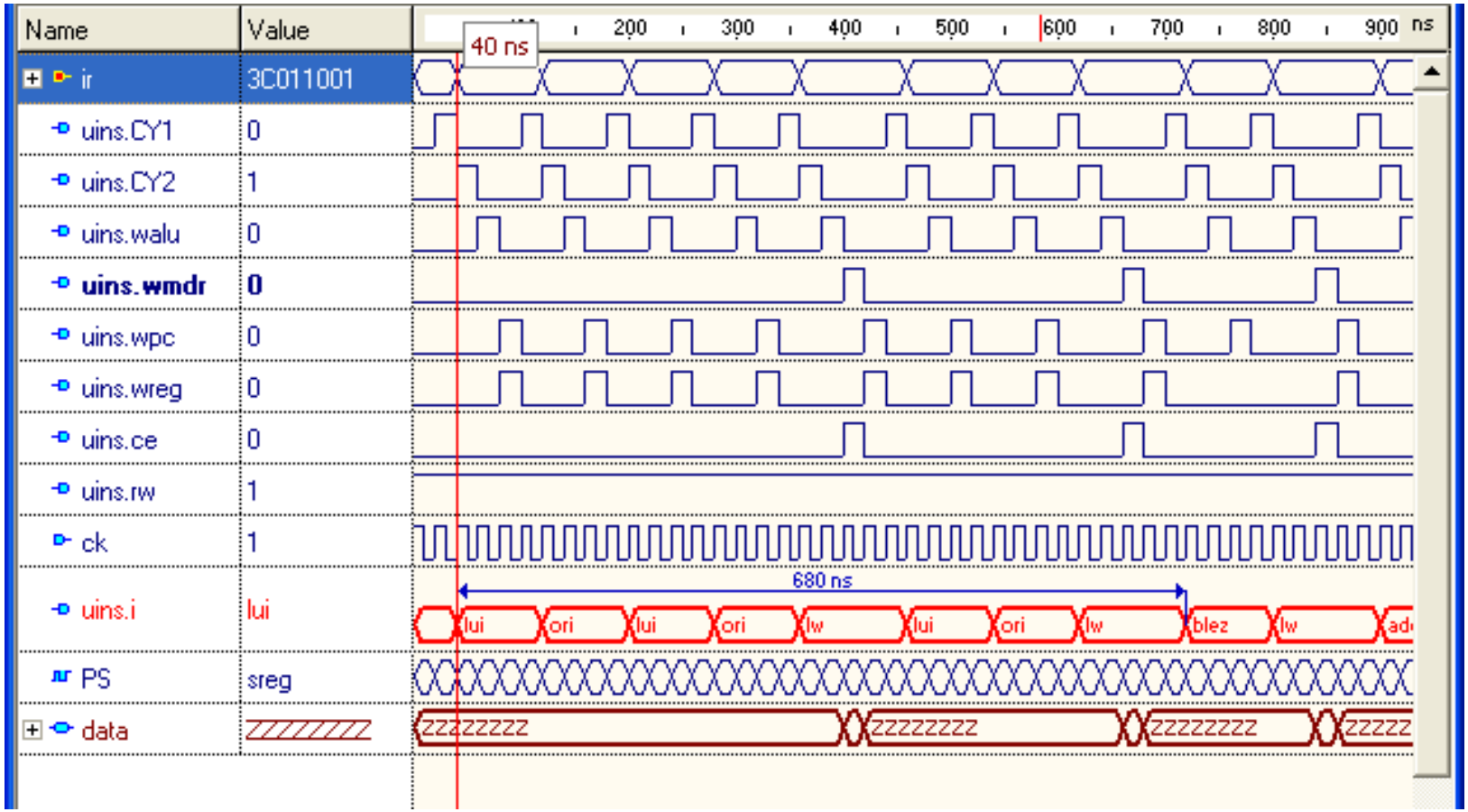
Resposta:
7,2 microsegundos

Fase de inicialização das variáveis

- 680 ns ➔ 34 ciclos de clock

- ANALISAR A MICROINSTRUÇÃO

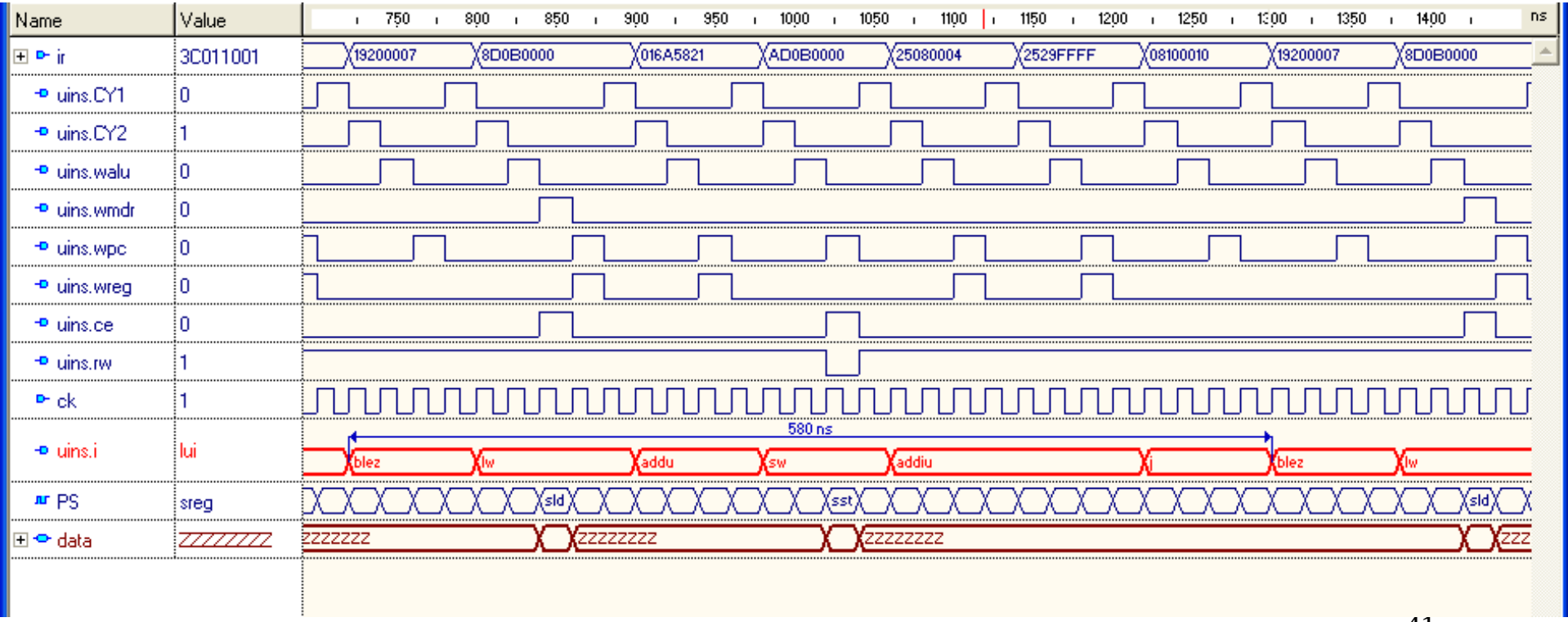
```
la    $t0,array
la    $t1,size
lw    $t1,0($t1)
la    $t2,const
lw    $t2,0($t2)
```



Uma iteração do laço

- 580 ns ➔ 29 ciclos de clock

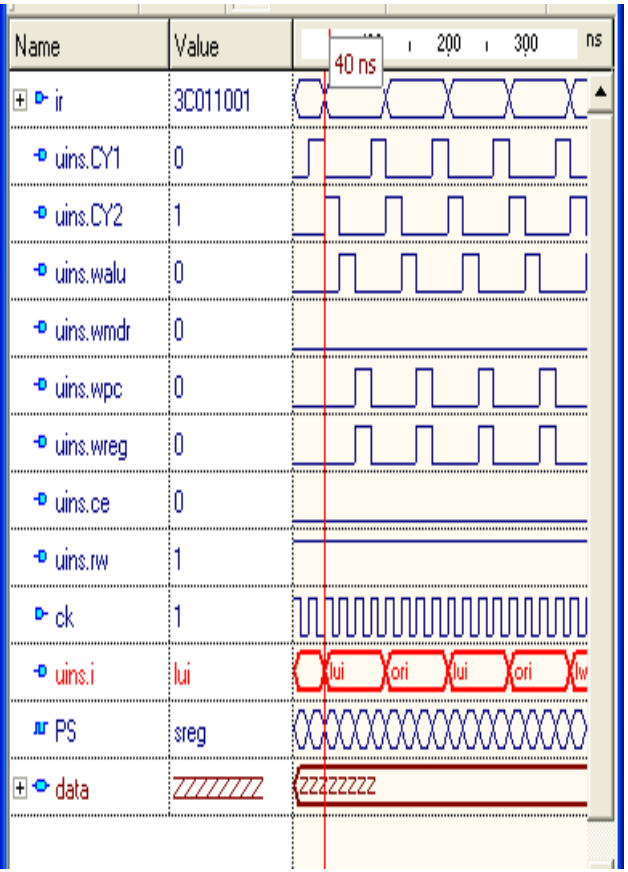
```
loop:  blez   $t1,end
       lw     $t3,0($t0)
       addu   $t3,$t3,$t2
       sw     $t3,0($t0)
       addiu  $t0,$t0,4
       addiu  $t1,$t1,-1
       j      loop
```



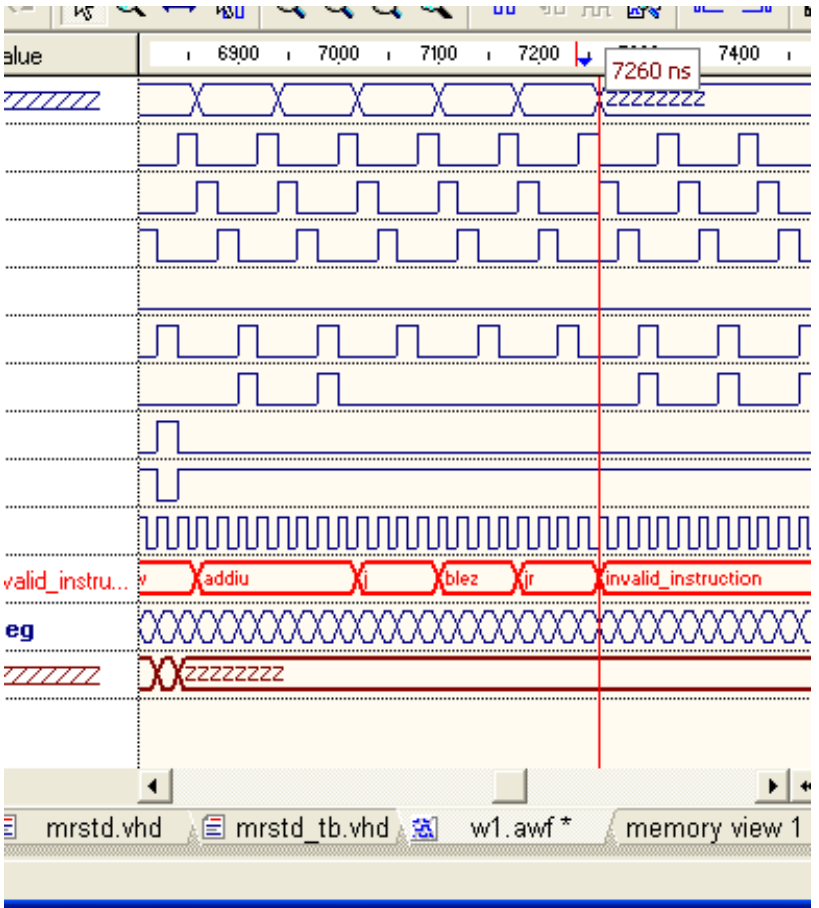
Tempo total de execução

- 7220 ns ➔ 361 ciclos de clock

- Início: 40 ns



- Início: 7260 ns



Exercício

- Cálculo do tempo de execução
  - Considerar que deseja-se utilizar o processador MIPS como um timer.
  - Frequência do processador: 10 MHz
  - Tempo do timer: 1 centésimo de segundo

  - **Pede-se**: escreva um laço que execute no tempo especificado pelo tempo do timer.