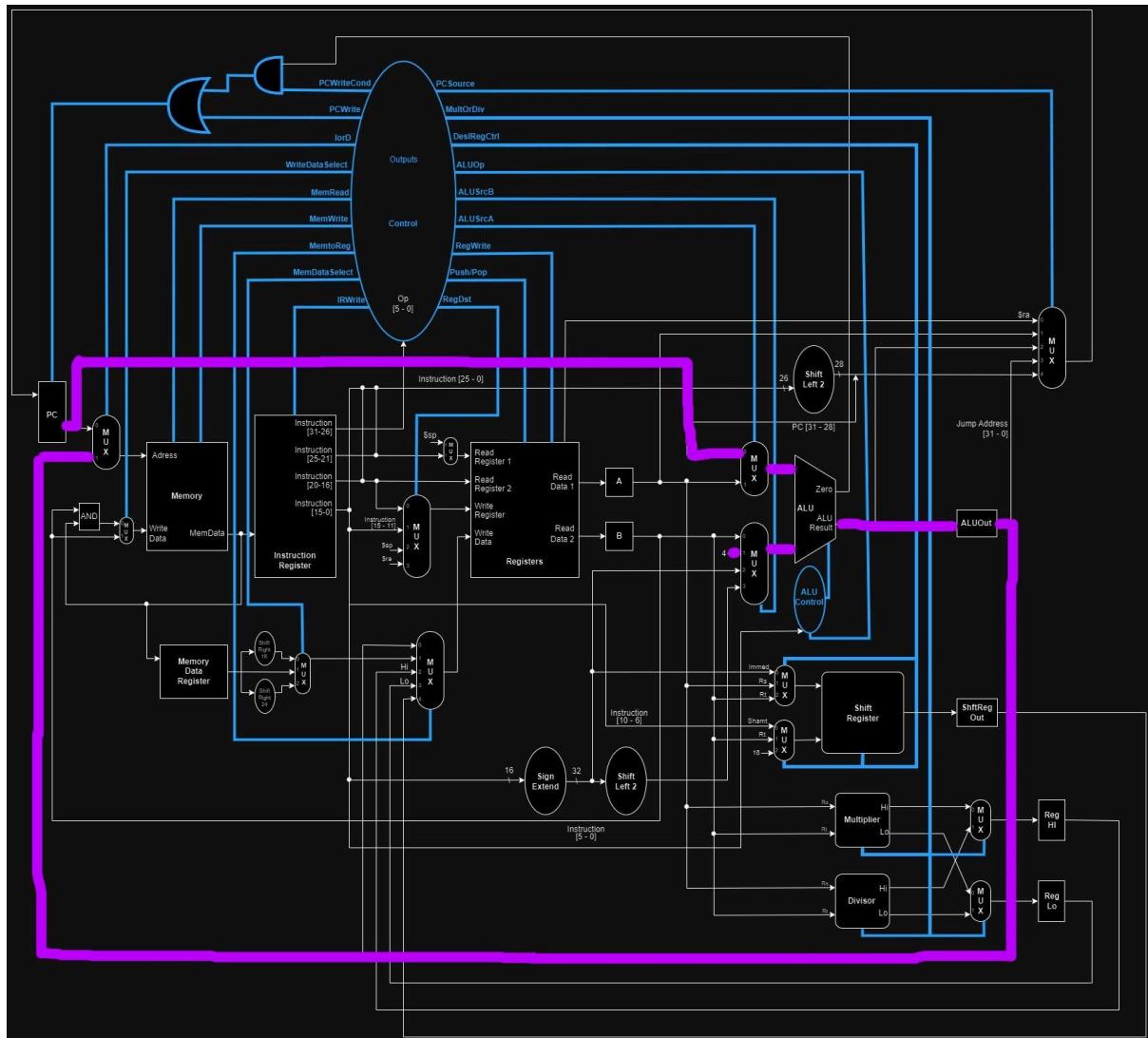


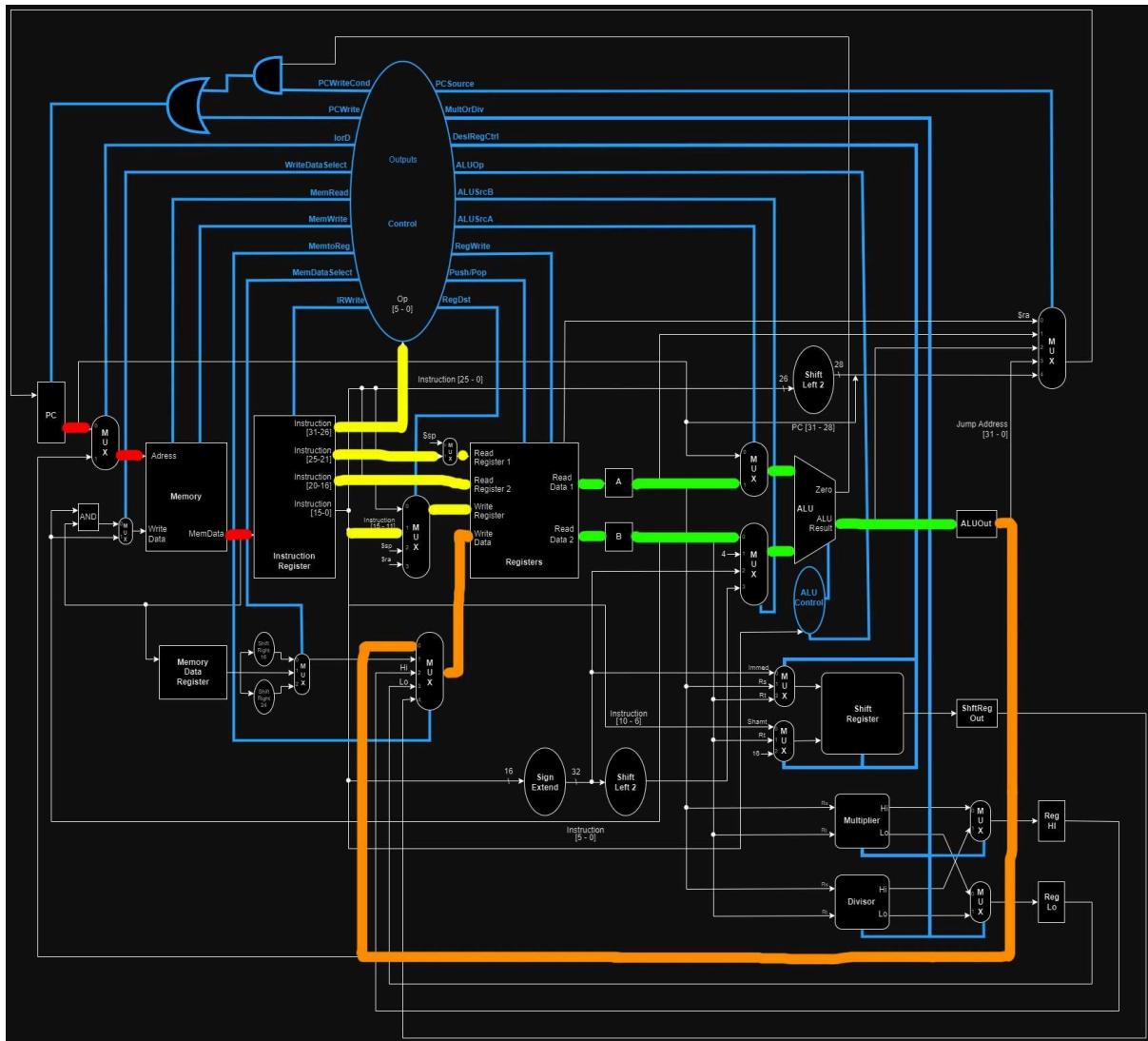
Incremento de PC (PC+4):



TIPO R

ADD (adição):

$$Rd \leftarrow rs + rt$$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e Rd.

(3) Verde - Execute:

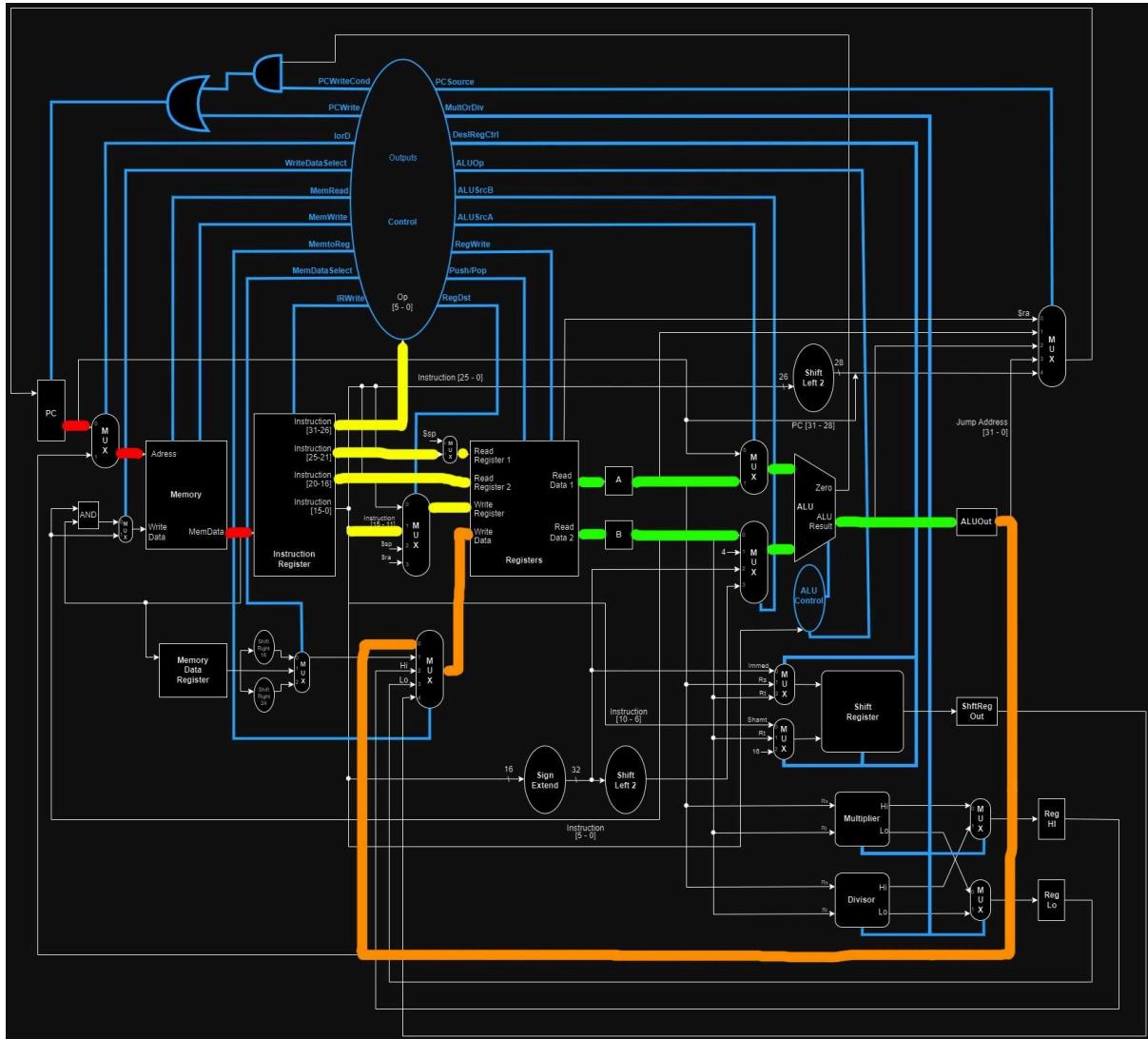
Executa a soma e deixa disponível na ALUOut.

(4) Laranja - Write Back:

Escreve o resultado de volta no registrador Rd.

AND (and lógico):

$Rd \leftarrow rs \& rt$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e Rd.

(3) Verde - Execute:

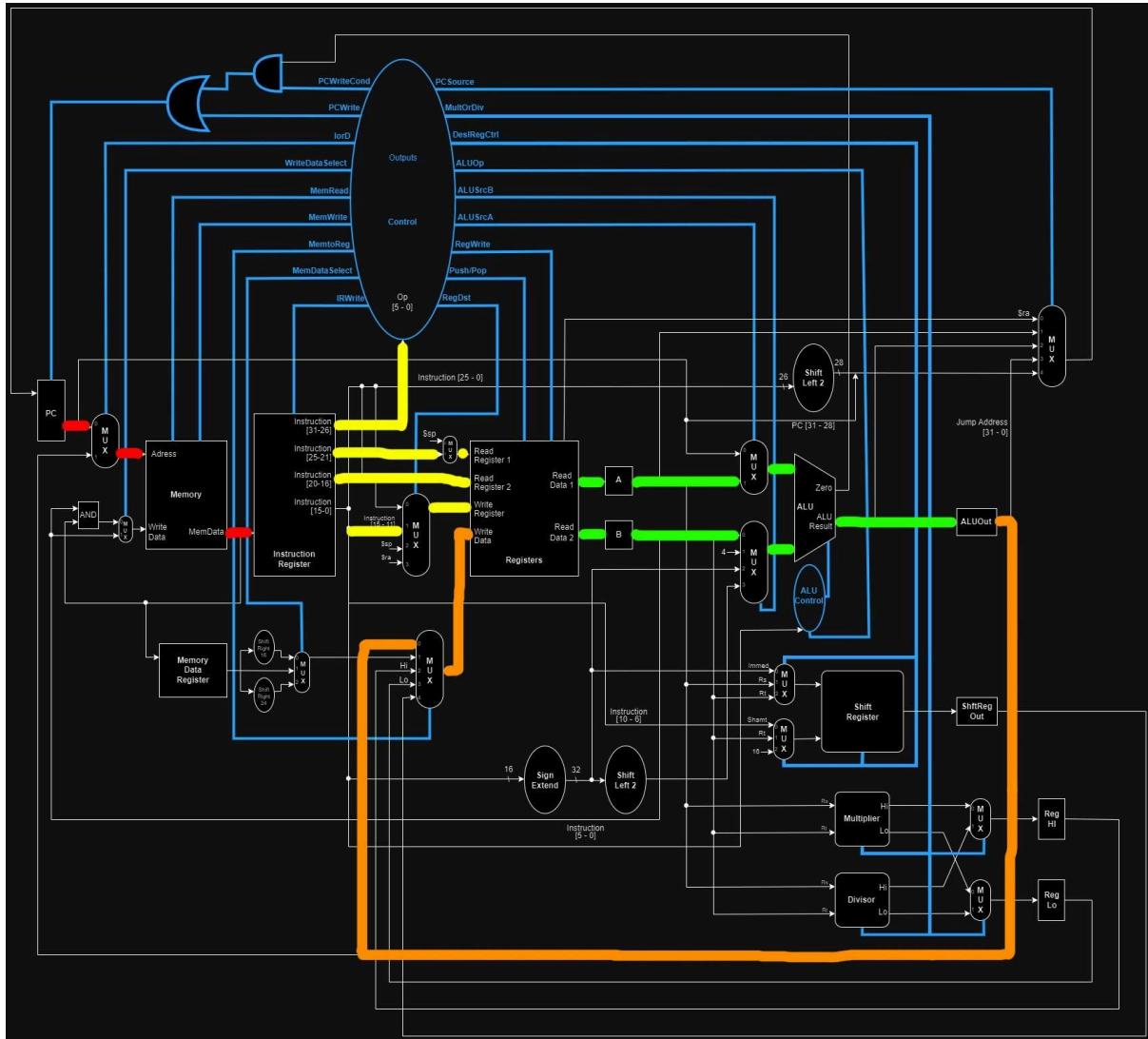
Executa a operação de AND e deixa disponível na ALUOut.

(4) Laranja - Write Back:

Escreve o resultado de volta no registrador Rd.

SUB (subtração):

$$Rd \leftarrow rs - rt$$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e Rd.

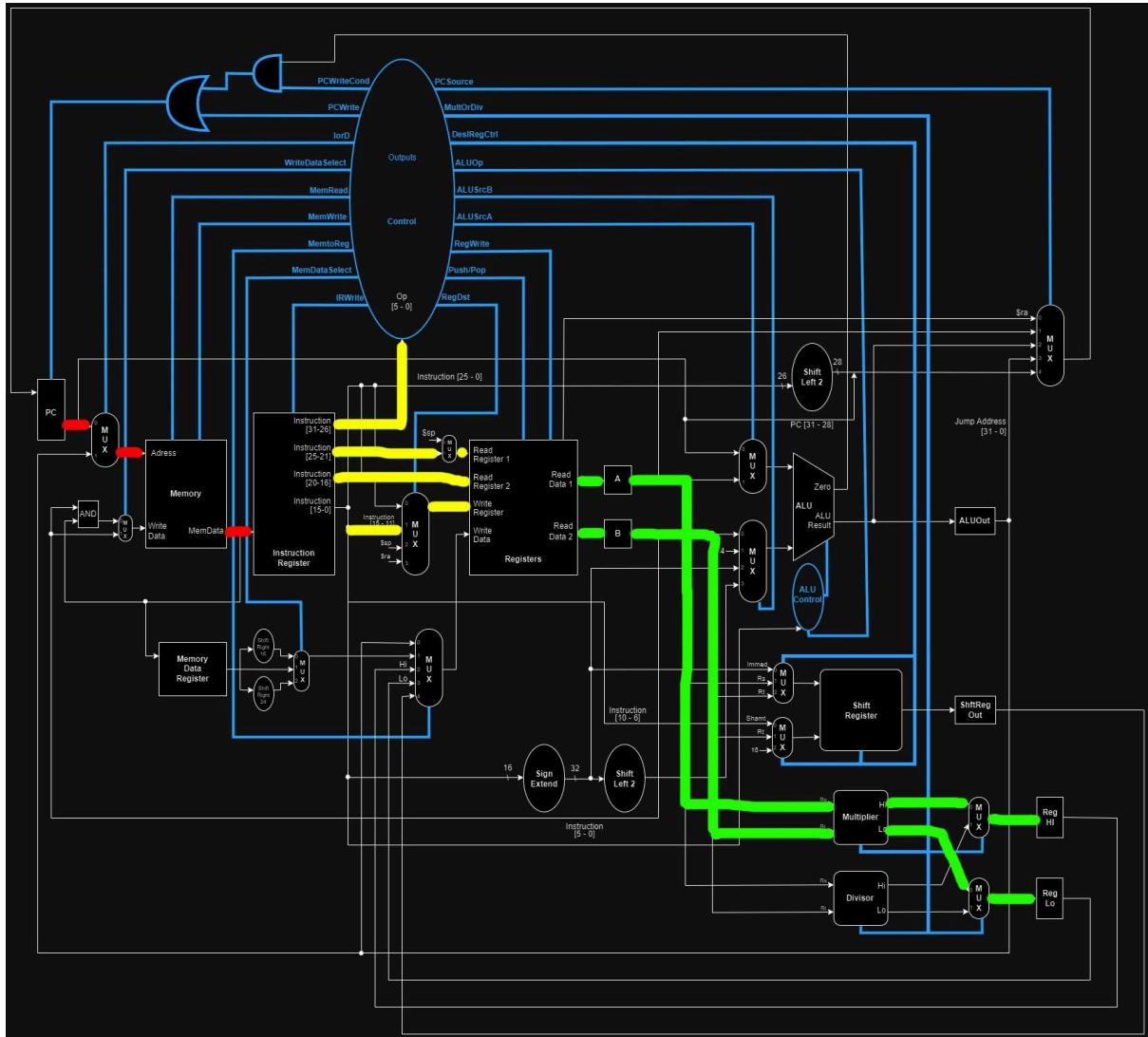
(3) Verde - Execute:

Executa a subtração e deixa disponível na ALUOut.

(4) Laranja - Write Back:

Escreve o resultado de volta no registrador Rd.

MULT (multiplicação):



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

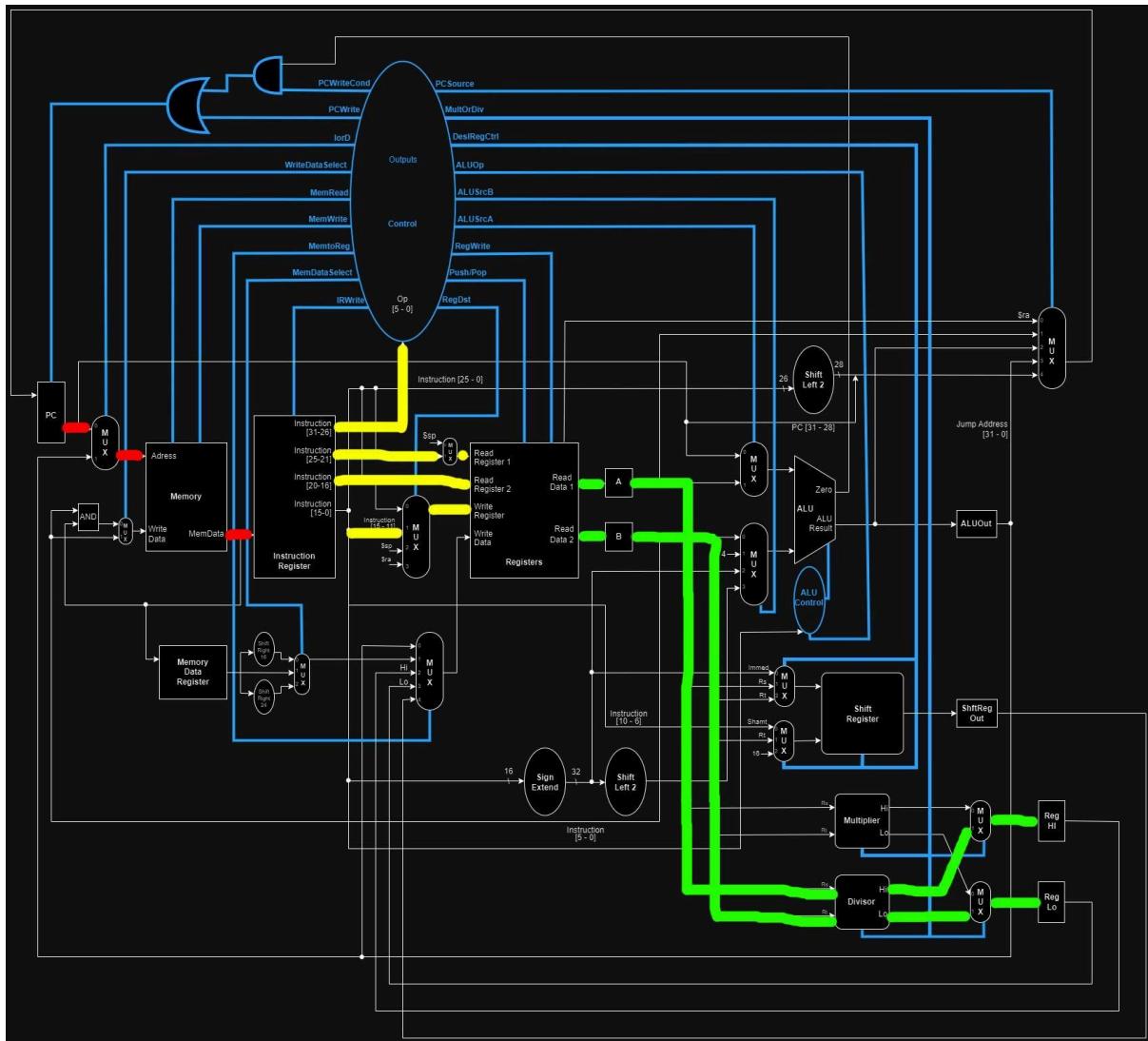
Lê o opcode e os registradores Rs, Rt e Rd.

(3) Verde - Execute:

Executa a operação de multiplicação.

Deixa os resultados armazenados em Hi e Lo.

DIV (divisão) :



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e Rd.

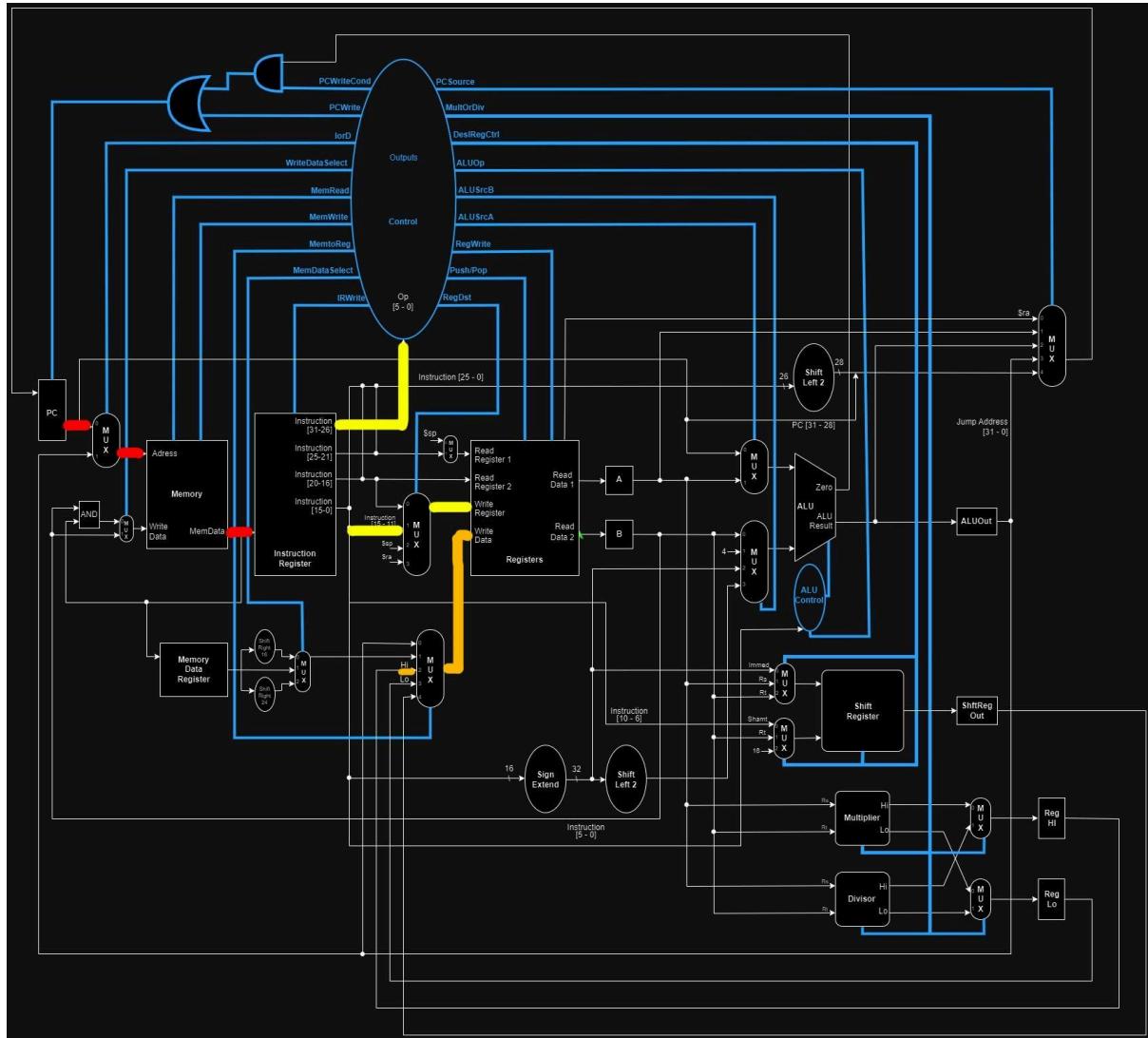
(3) Verde - Execute:

Executa a operação de divisão.

Deixa os resultados armazenados em Hi e Lo.

MFHI (move from HI):

$rd \leftarrow hi$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

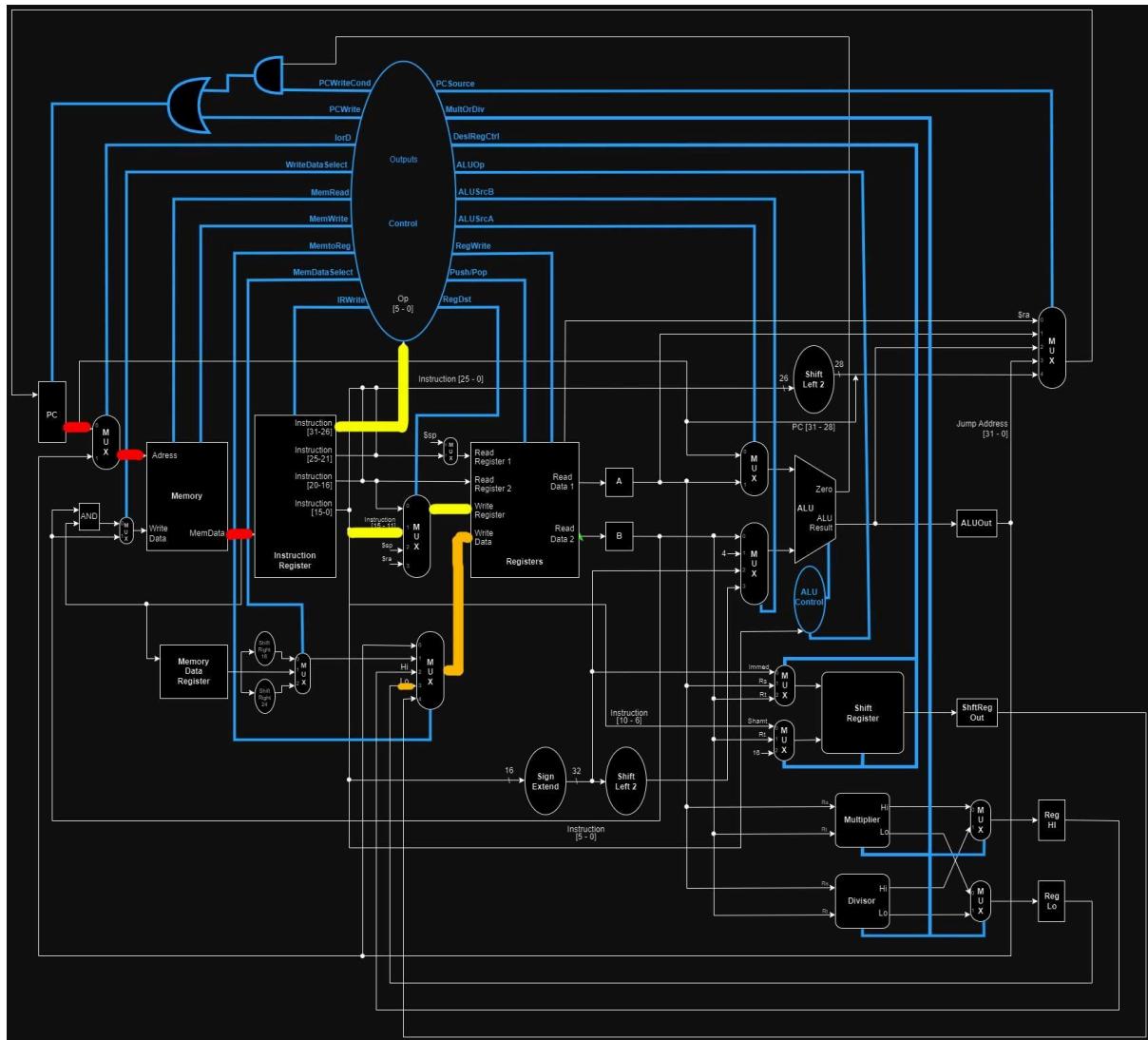
Lê o opcode e o registrador Rd.

(3) Laranja - Write Back:

Escreve o valor de Hi em Rd.

MFLO (move from LO):

$rd \leftarrow lo$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

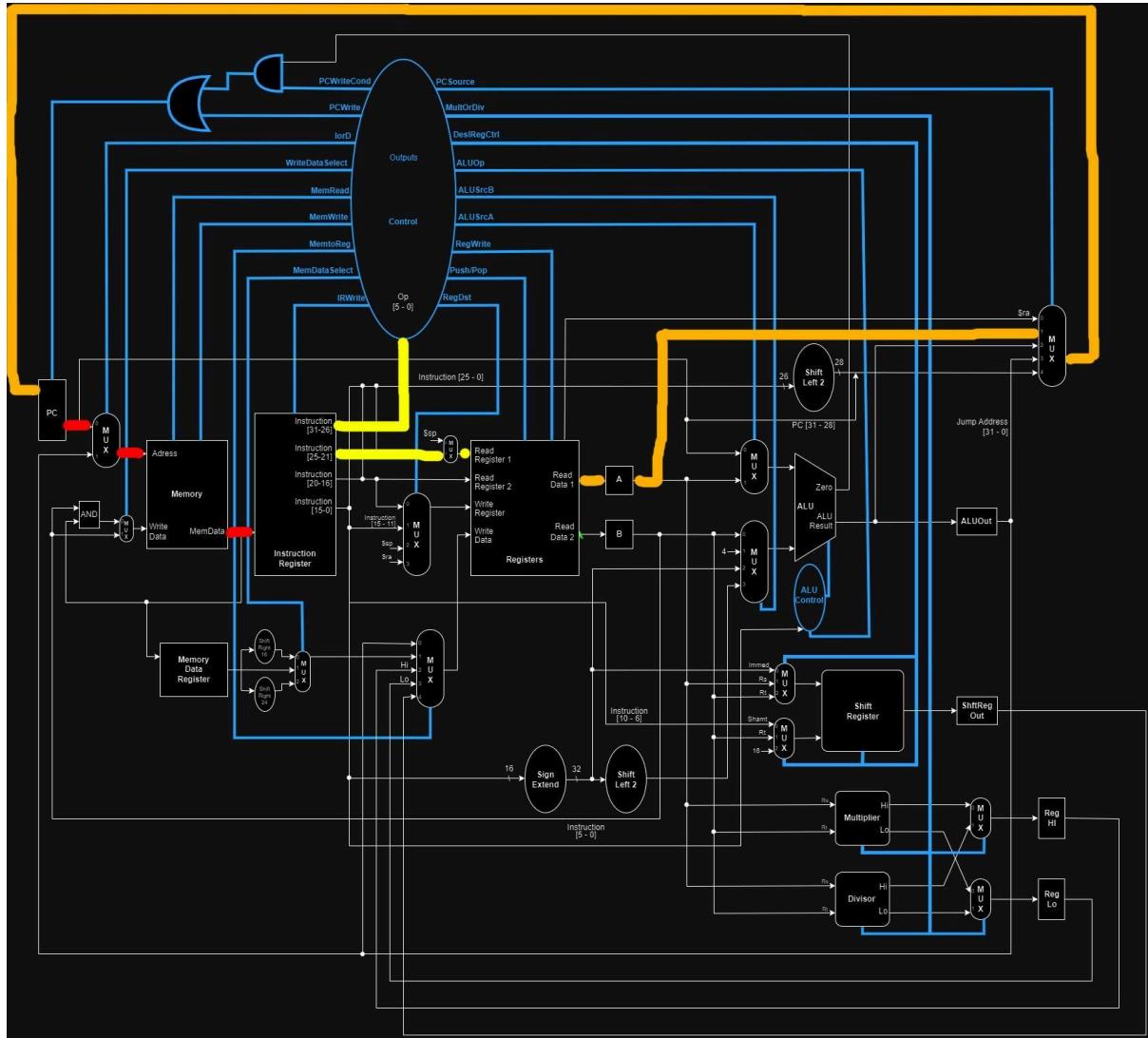
Lê o opcode e o registrador Rd.

(3) Laranja - Write Back:

Escreve o valor de Lo em Rd.

JR (jump register):

$PC \leftarrow rs$



(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

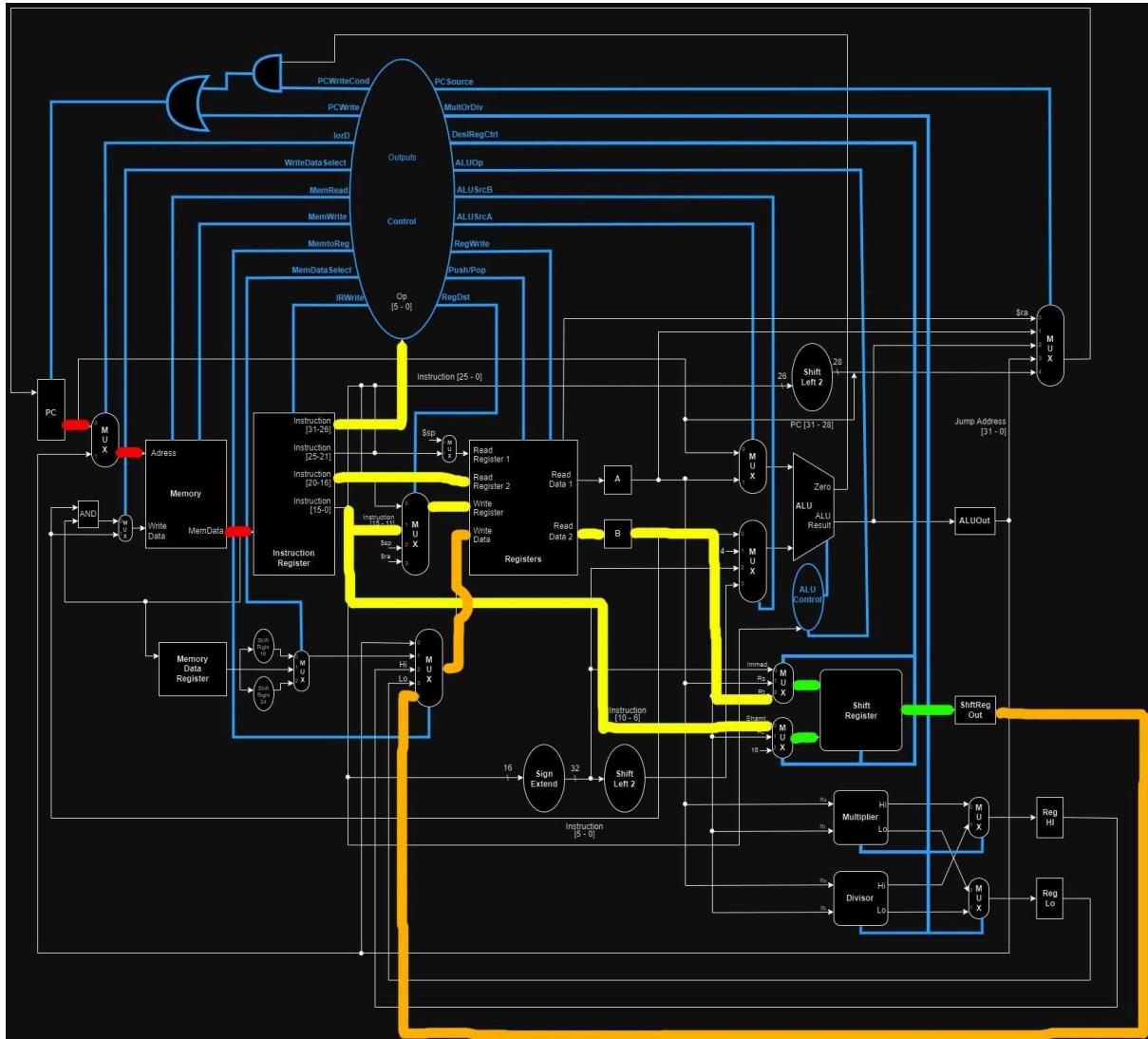
Lê o opcode e o registrador Rs.

(3) Laranja - Write Back:

Escreve o valor de \$Rs no PC.

SLL (shift left logical):

$rd \leftarrow rt \ll shamt$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rt, Shamt e Rd.

(3) Verde - Execute:

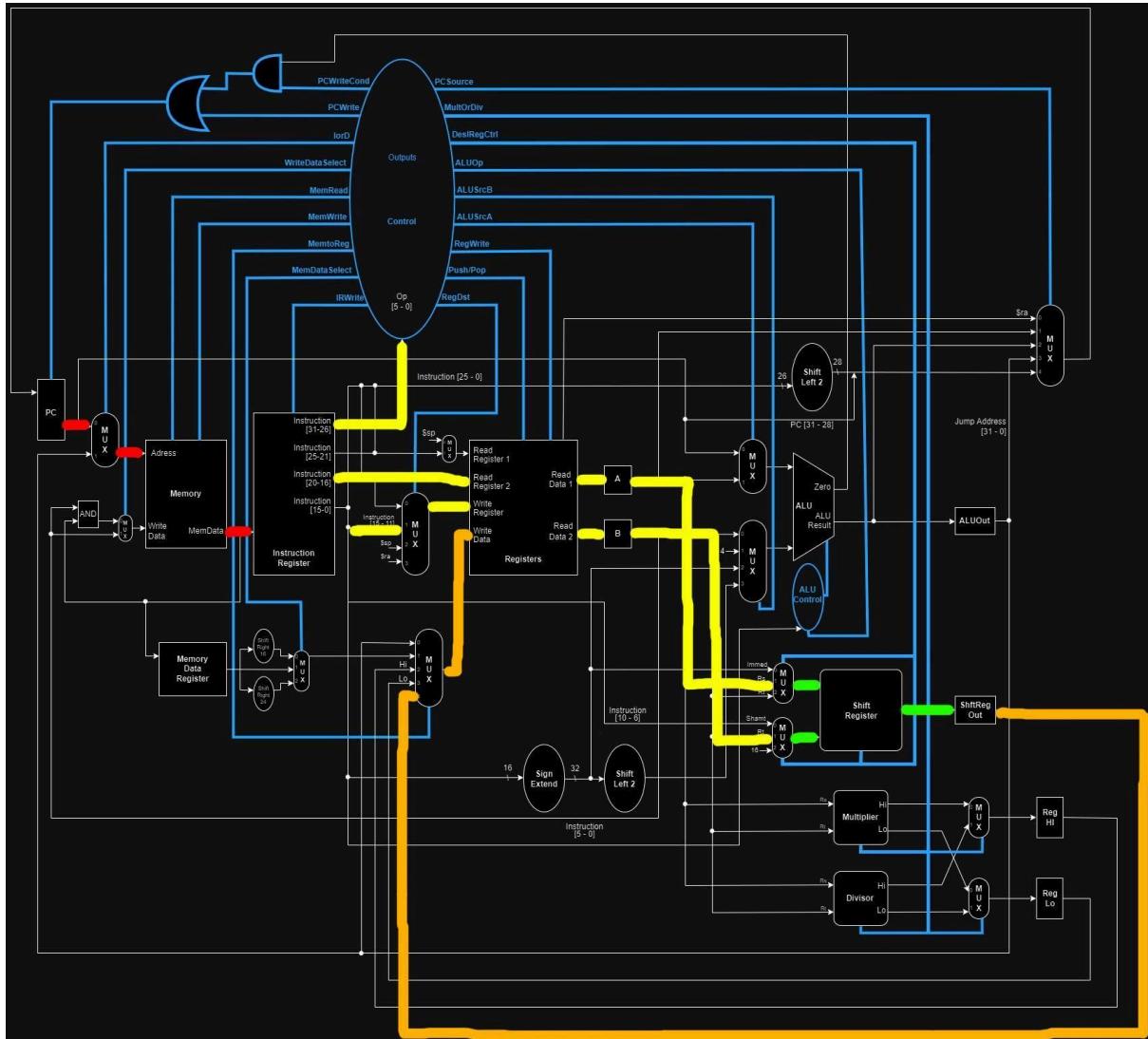
Executa o shift ($Rt \ll Shamt$) e deixa disponível no ShiftRegOut.

(4) Laranja - Write Back:

Escreve o resultado de volta no registrador Rd.

SLLV (shift left logical variable):

$$rd \leftarrow rs \ll rt$$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e Rd.

(3) Verde - Execute:

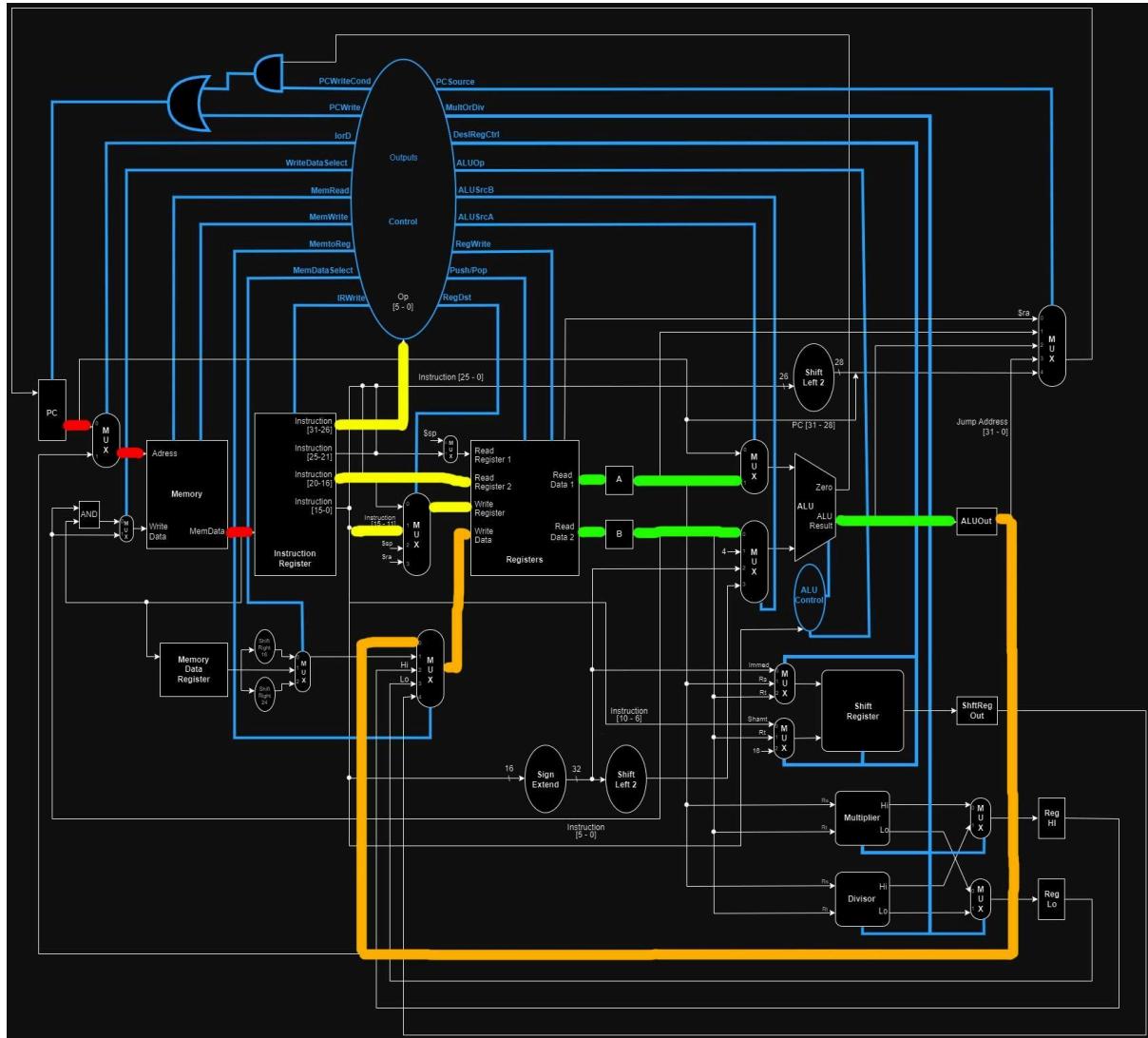
Executa o shift ($Rs \ll Rt$) e deixa disponível no ShiftRegOut.

(4) Laranja - Write Back:

Escreve o resultado de volta no registrador Rd.

SLT (set less than):

$$rd \leftarrow (rs < rt) ? 1 : 0$$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e Rd.

(3) Verde - Execute:

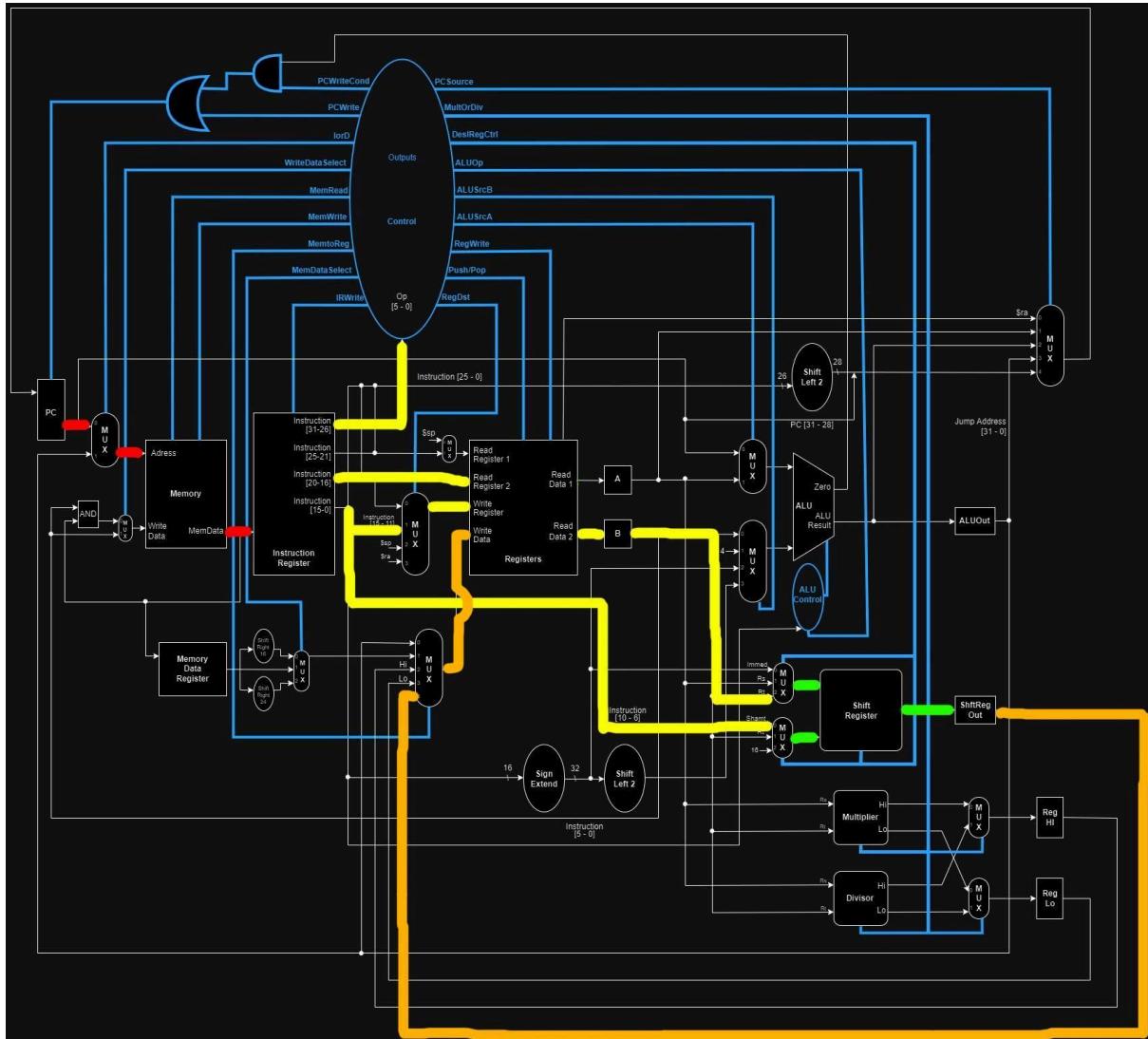
Compara os valores (Rs < Rt).

Resultado 0 ou 1 fica disponível em ALUOut.

(4) Laranja - Write Back:

Escreve o resultado de volta no registrador Rd.

SRL (shift right logical): $rd \leftarrow rt >> shamt$



(0) Incremento de PC

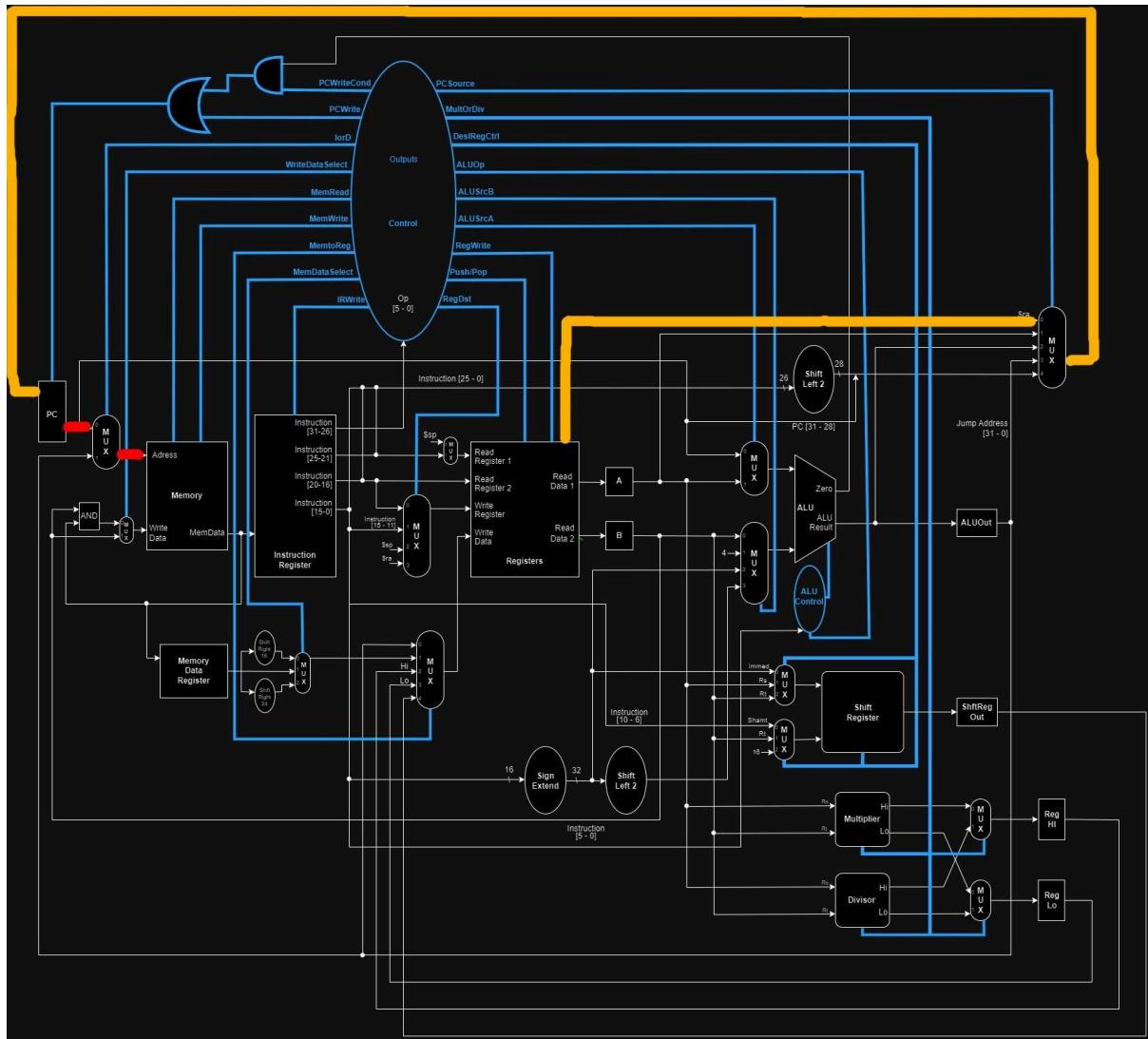
(1) Vermelho - Instruction Fetch:
 Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:
 Lê o opcode e os registradores Rt, Shamt e Rd.

(3) Verde - Execute:
 Executa o shift ($Rt >> Shamt$) e deixa disponível no ShiftRegOut.

(4) Laranja - Write Back:
 Escreve o resultado de volta no registrador Rd.

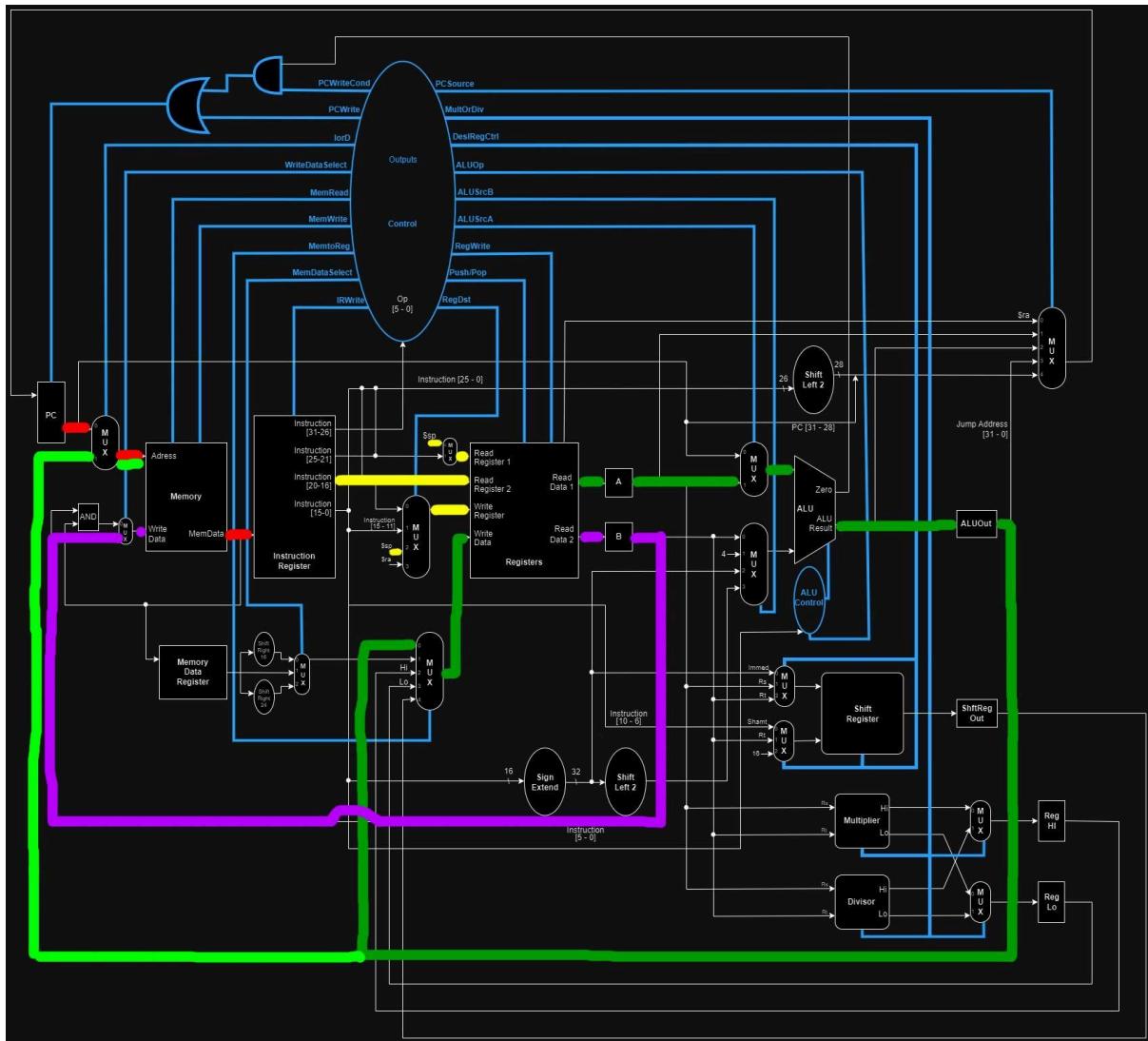
RTE (return from exception): PC EPC



(1) Vermelho - Instruction Fetch:
Busca a instrução na memória de instruções.

(2) Laranja - Write Back:
Escreve o valor de \$Ra no PC.

PUSH (push para pilha):



(0) *Incremento de PC*

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e o registrador Rt.

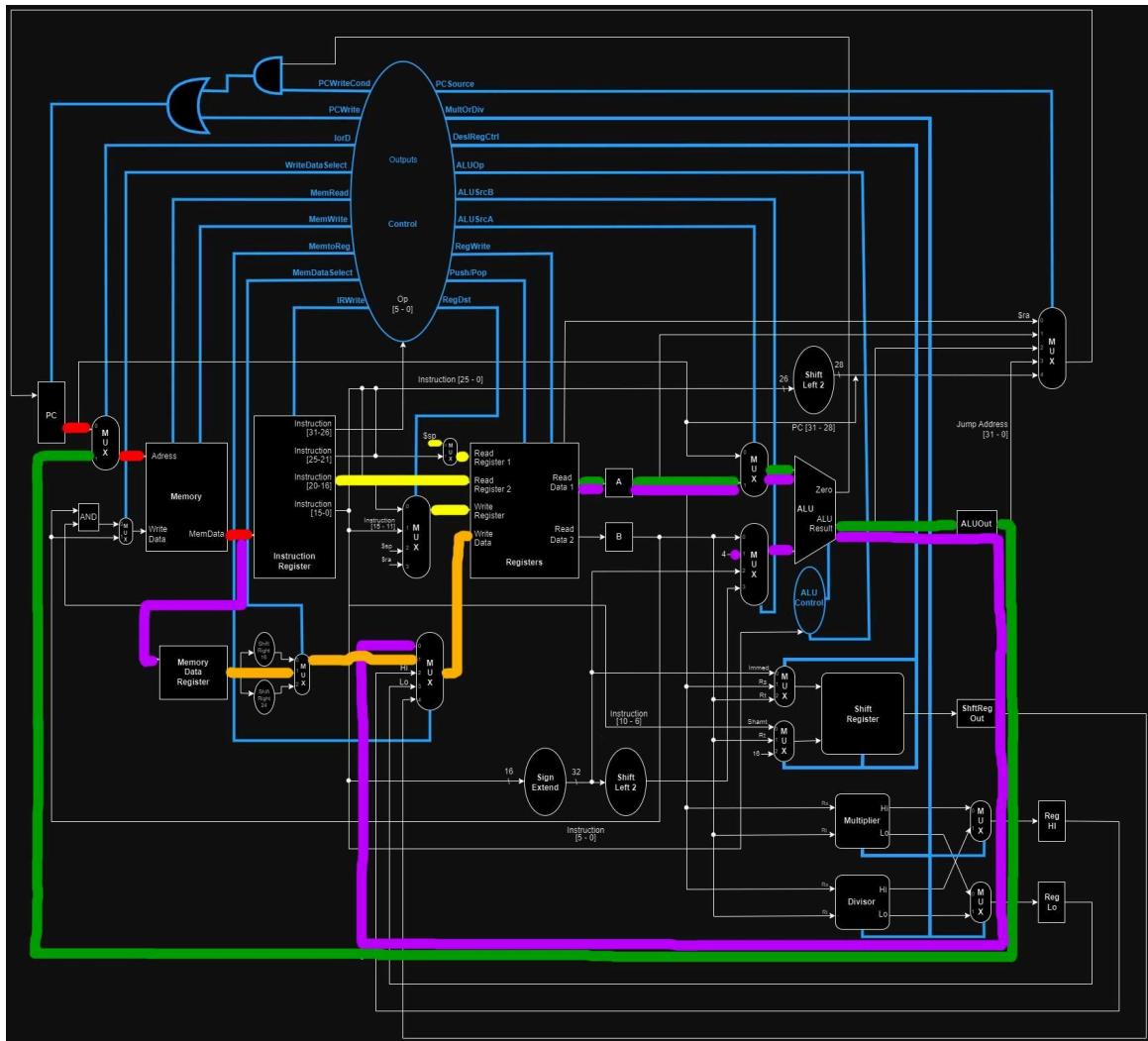
(3) Verde - Execute:

(Verde escuro) = Decrementa o ponteiro do topo da pilha no banco de registradores
 (Verde claro) = Encaminha o endereço o qual será escrito o dado (\$sp - 4).

(4) Roxo - Memory Access:

Escreve o valor de Rt no topo da pilha.

POP (pop na pilha):



(0) *Incremento de PC*

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e o registrador Rt.

(3) Verde - Execute:

Encaminha o endereço correspondente ao topo da pilha (\$sp).

(4) Roxo - Memory Access:

Obtém o valor armazenado em \$sp e guarda em Memory Data Register. Incrementa o ponteiro da pilha (\$sp + 4).

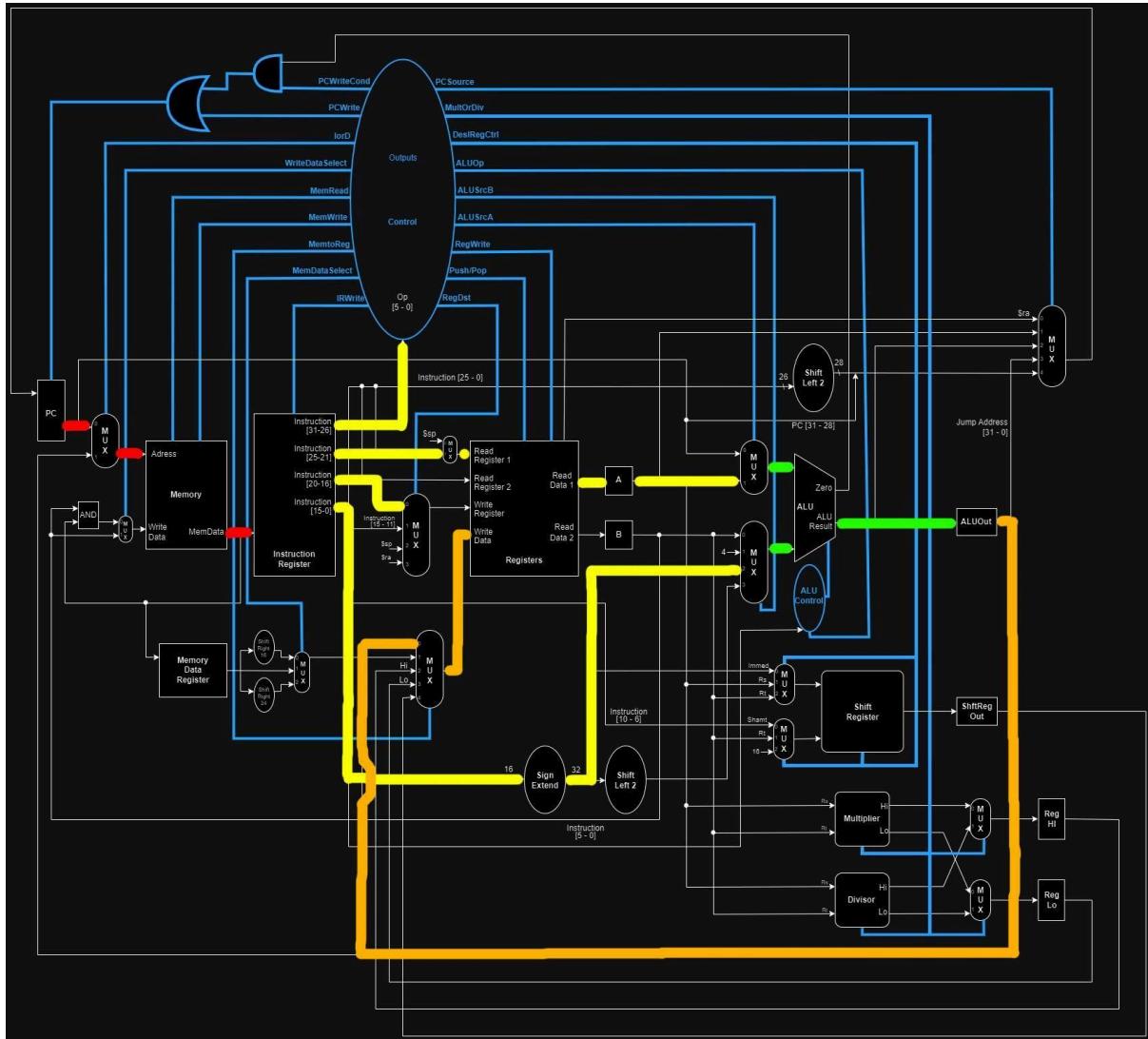
(5) Laranja - Write Back:

Escreve o valor obtido do topo da pilha em Rt.

TIPO I

ADDI (adição imediata):

$Rt \leftarrow rs + \text{immediato}$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Immediato e Rd.

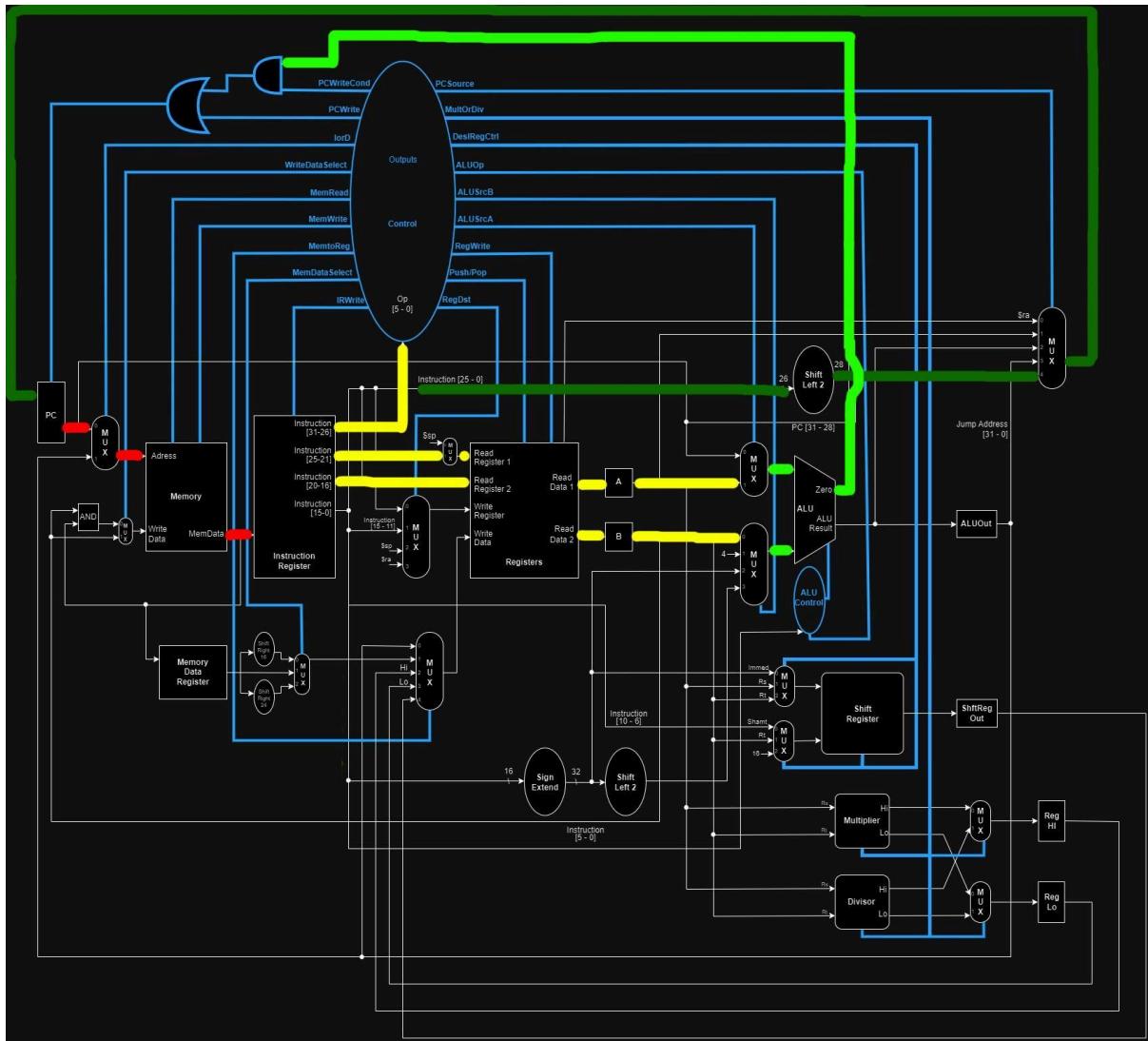
(3) Verde - Execute:

Executa a soma e deixa disponível na ALUOut.

(4) Laranja - Write Back:

Escreve o resultado de volta no registrador Rd.

BEQ (branch if equal):



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs e Rt.

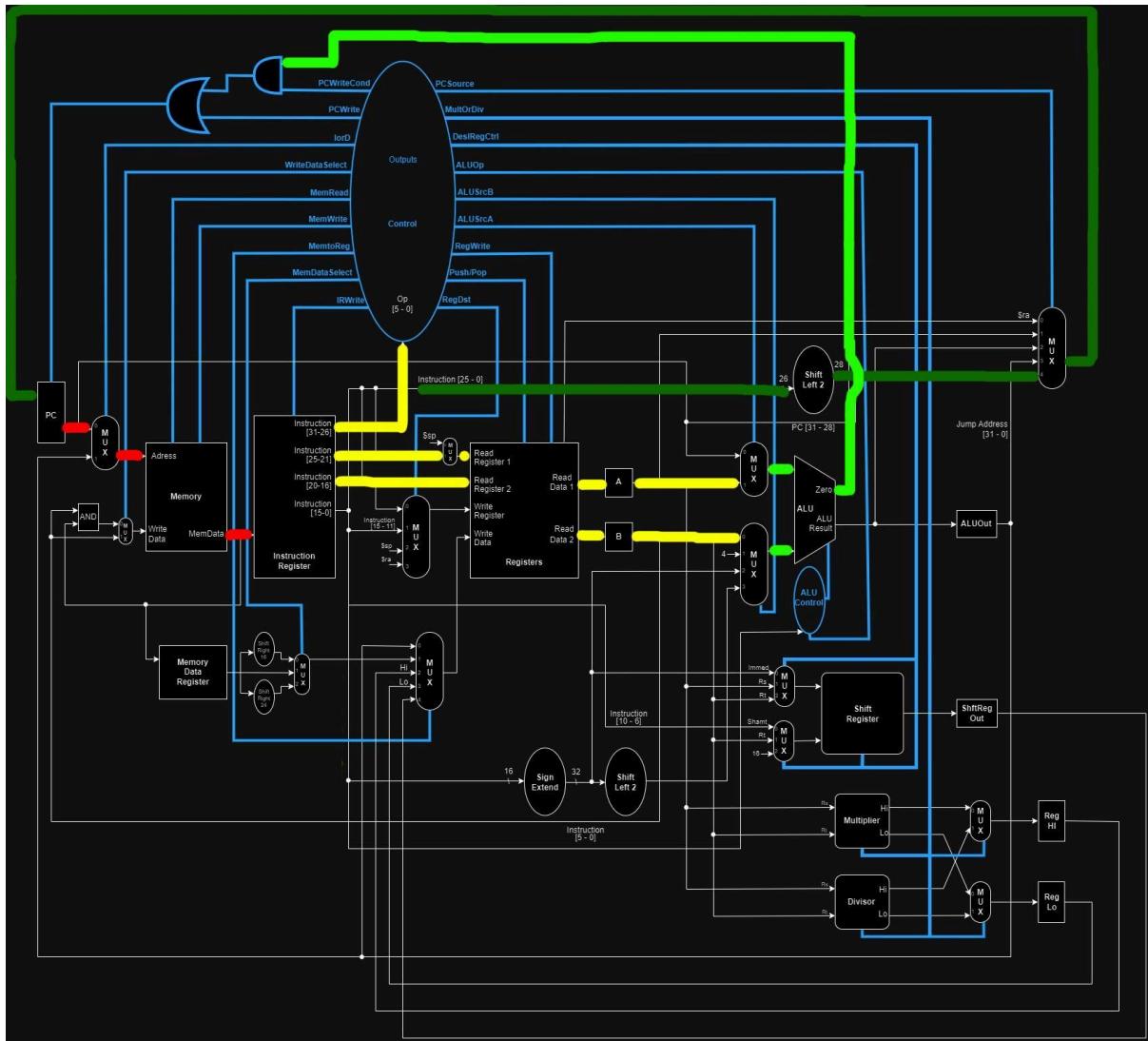
(3) Verde - Execute:

(Verde claro) = Executa a comparação ($Rs == Rt$) e a ULA emite o status (1 ou 0).

(Verde escura) = Realiza ($PC + 4 + (\text{offset} \ll 2)$).

Se verdadeiro, o novo endereço é encaminhado para PC.

BNE (branch not equal):



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

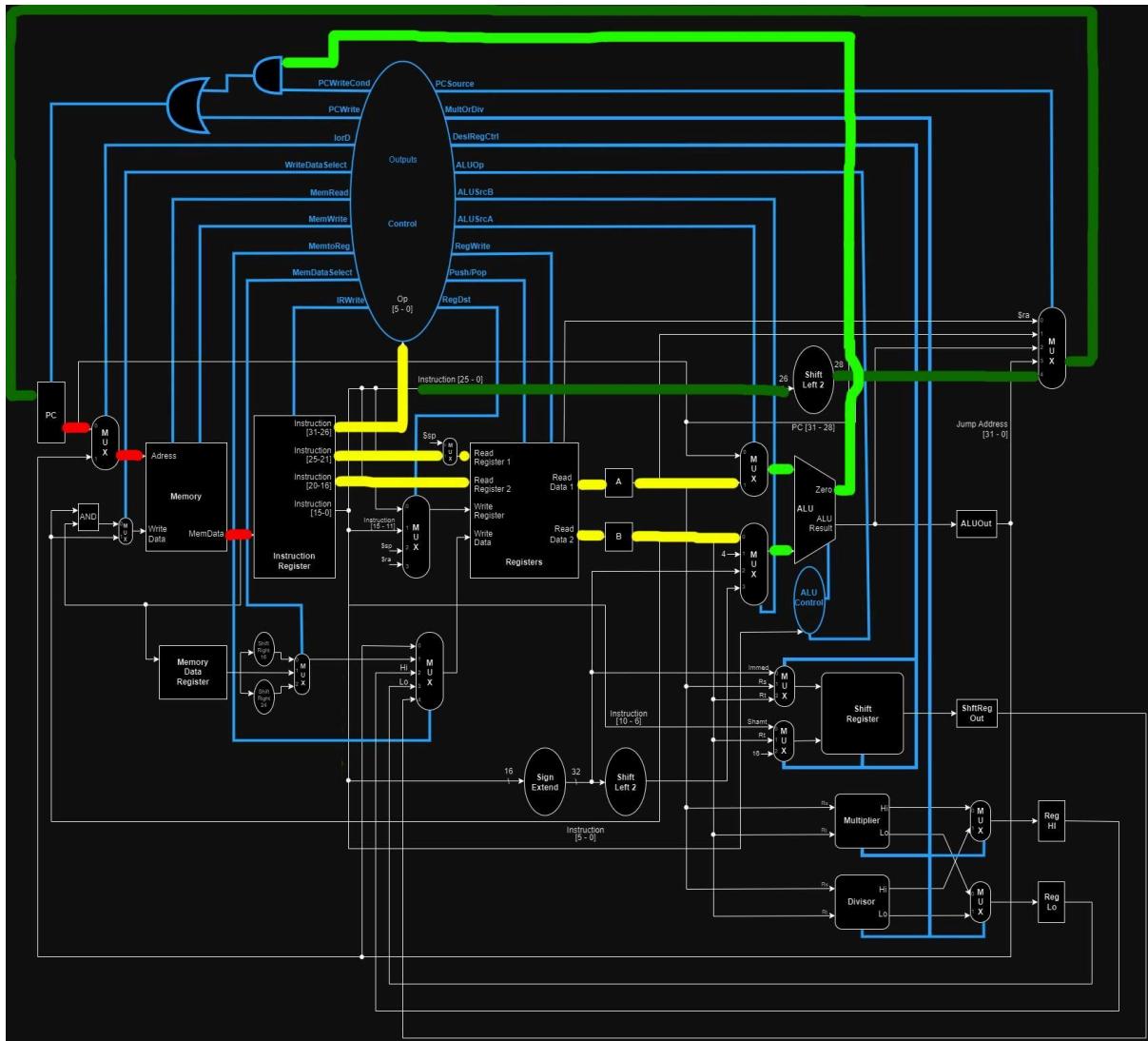
Lê o opcode e os registradores Rs e Rt.

(3) Verde - Execute:

(Verde claro) = Executa a comparação ($Rs \neq Rt$) e a ULA emite o status (1 ou 0).
 (Verde escura) = Realiza ($PC + 4 + (\text{offset} \ll 2)$).

Se verdadeiro, o novo endereço é encaminhado para PC.

BLE (branch if less than or equal to):



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs e Rt.

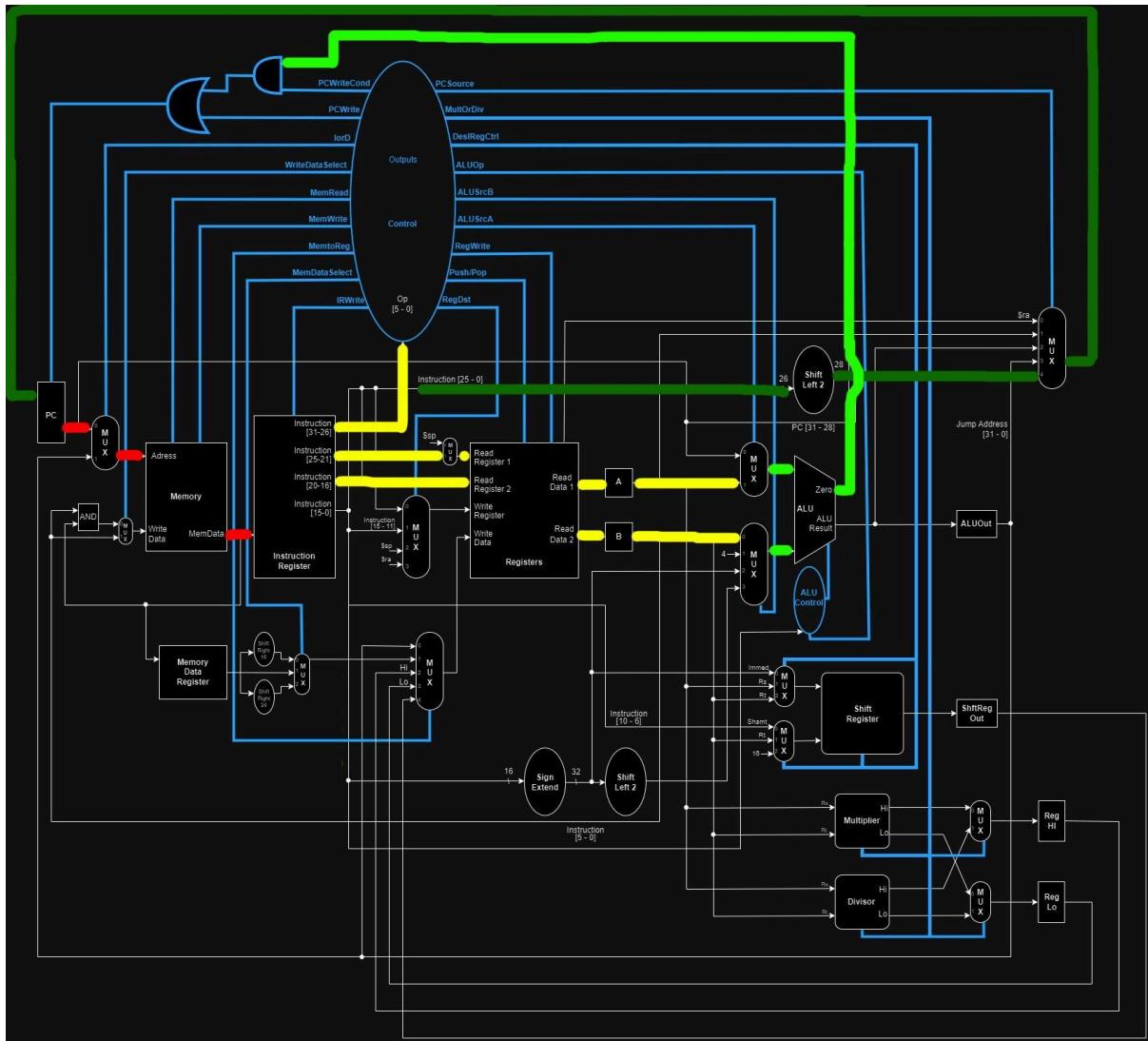
(3) Verde - Execute:

(Verde claro) = Executa a comparação ($Rs \leq Rt$) e a ULA emite o status (1 ou 0).

(Verde escura) = Realiza ($PC + 4 + (\text{offset} \ll 2)$).

Se verdadeiro, o novo endereço é encaminhado para PC.

BGT (branch if greater than):



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs e Rt.

(3) Verde - Execute:

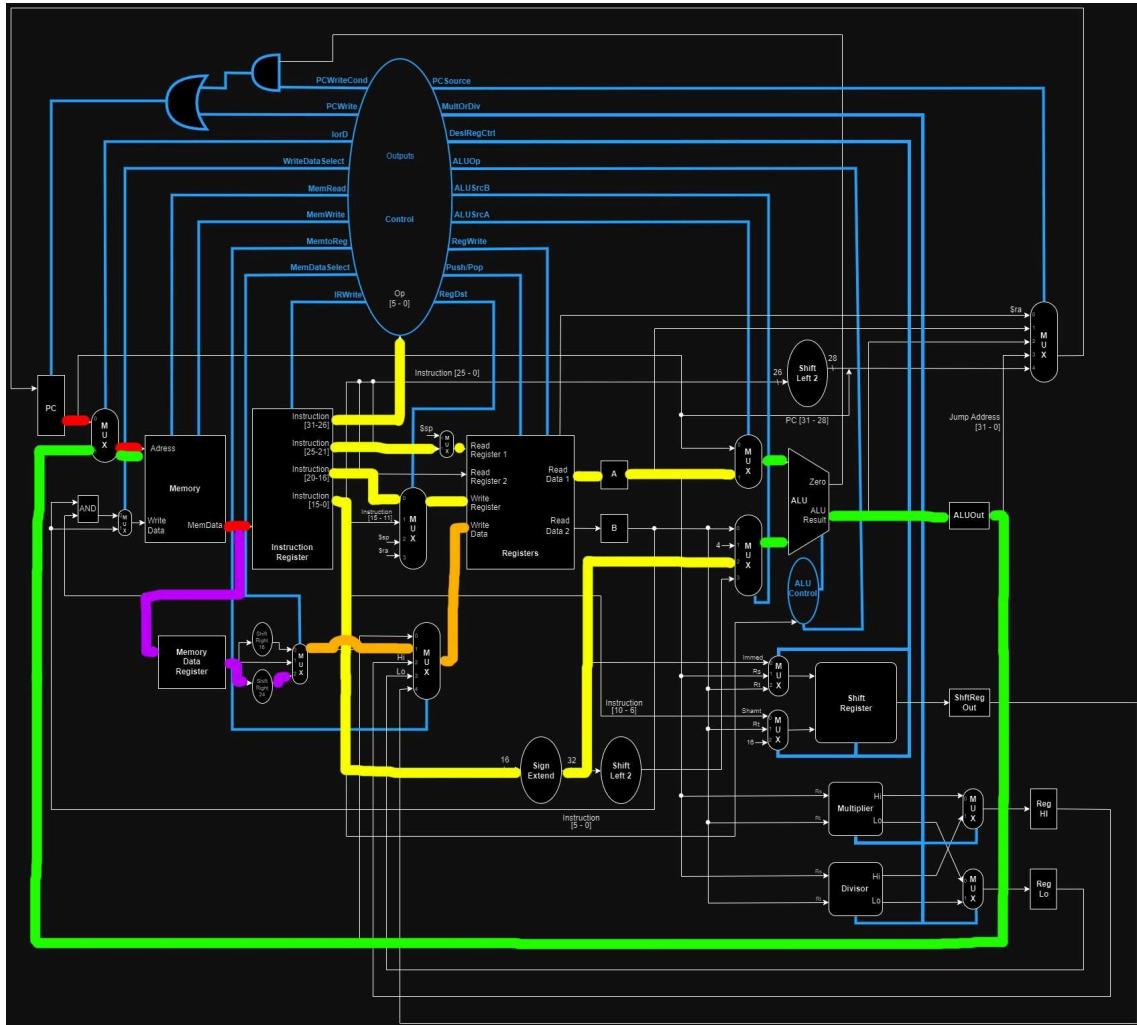
(Verde claro) = Executa a comparação ($Rs > Rt$) e a ULA emite o status (1 ou 0).

(Verde escura) = Realiza ($PC + 4 + (\text{offset} \ll 2)$).

Se verdadeiro, o novo endereço é encaminhado para PC.

LB (load byte):

$$Rt \leftarrow \text{byte Mem}[offset + rs]$$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e offset.

(3) Verde - Execute:

Calcula o endereço do byte (Rs + offset).

(4) Roxo - Memory Access:

Obtém a palavra (4 bytes) que existe nesse endereço base.

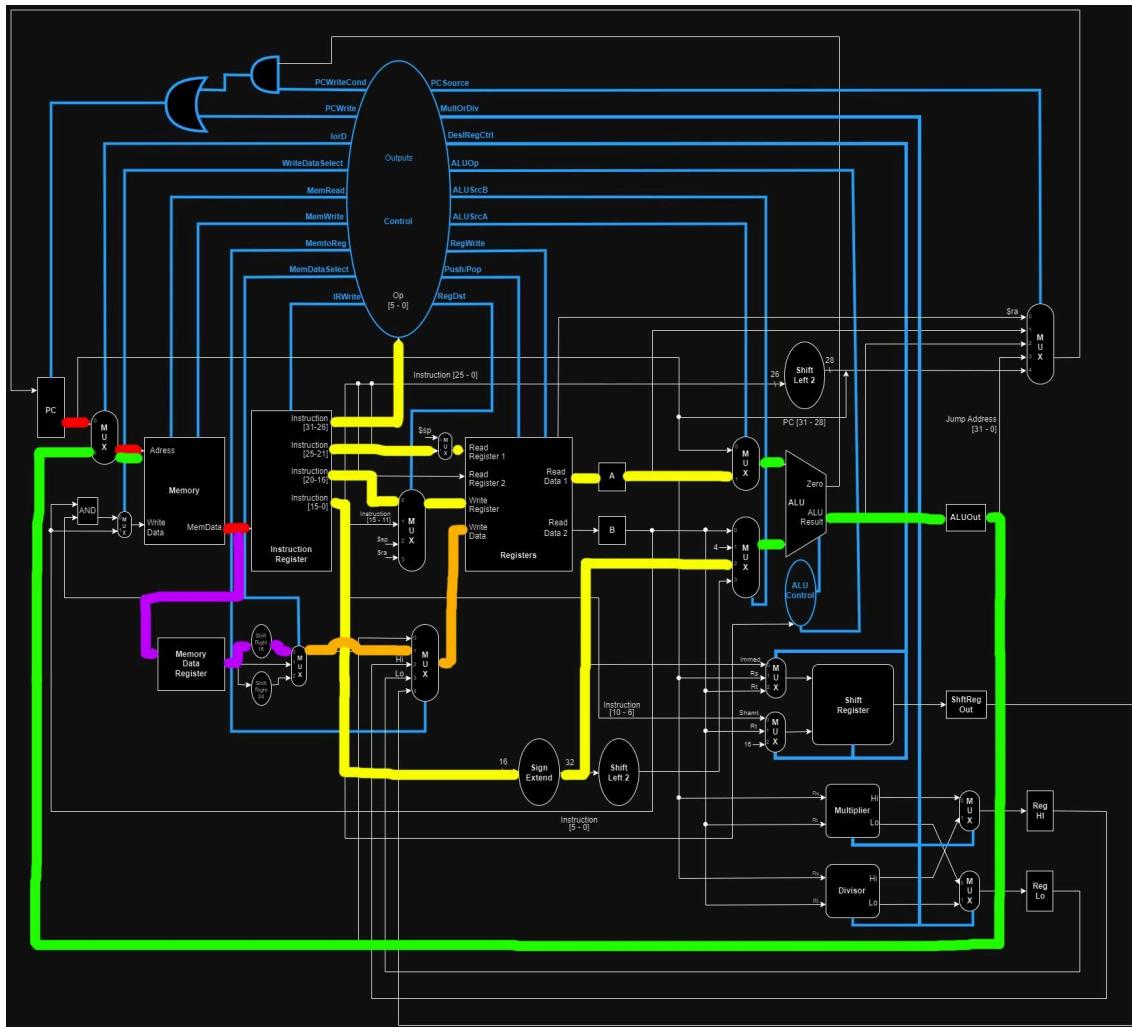
Faz um shift right 24 para obter apenas o 1 byte desejado dos 4 bytes da palavra.

(5) Laranja - Write Back:

Escreve o valor obtido em Rt.

LH (load halfword):

$Rt \leftarrow \text{halfword Mem}[offset + rs]$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e offset.

(3) Verde - Execute:

Calcula o endereço do byte (Rs + offset).

(4) Roxo - Memory Access:

Obtém a palavra (4 bytes) que existe nesse endereço base.

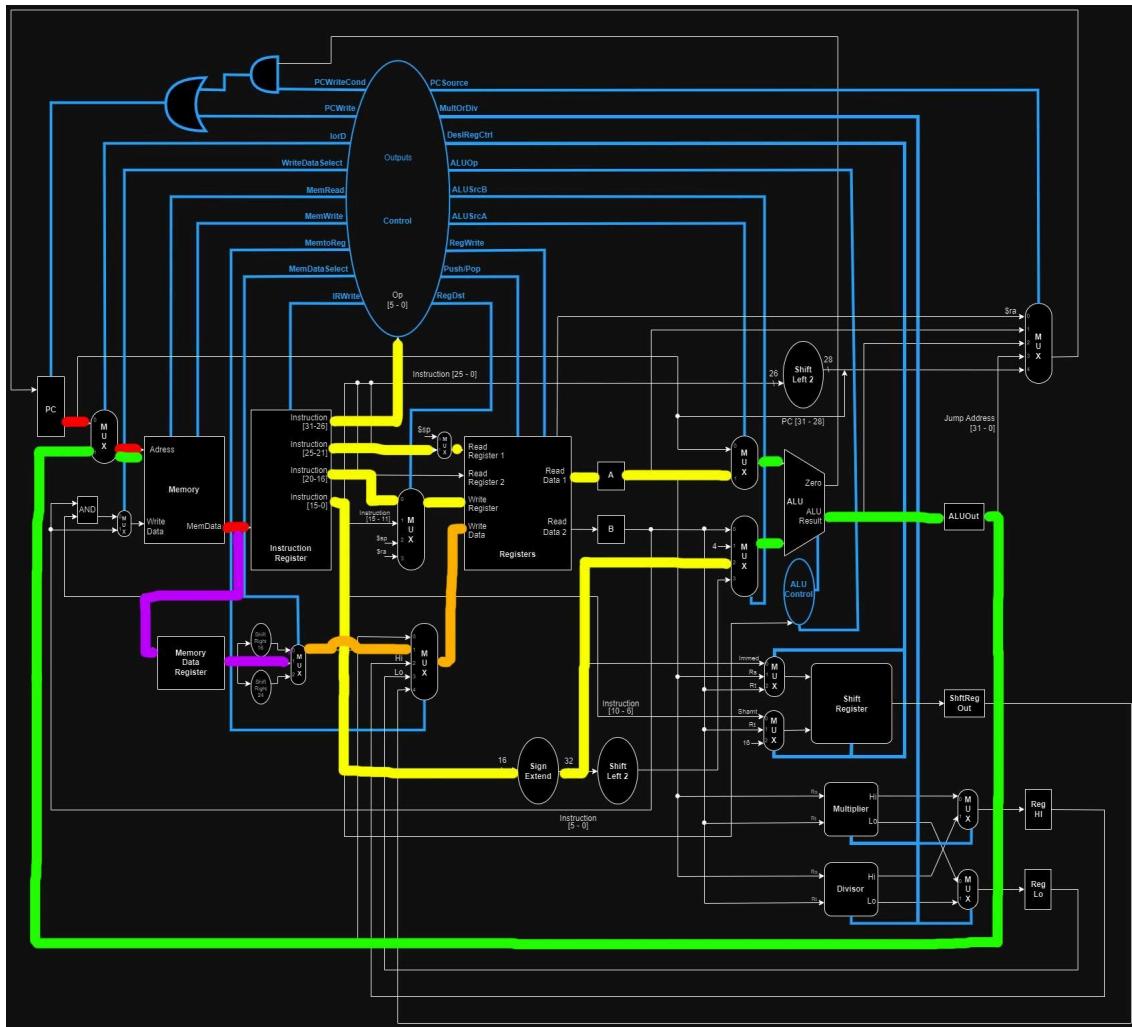
Faz um shift right 16 para obter apenas os 2 bytes desejado dos 4 bytes da palavra.

(5) Laranja - Write Back:

Escreve o valor obtido em Rt.

LW (load word):

$Rt \leftarrow \text{Mem}[offset + rs]$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e offset.

(3) Verde - Execute:

Calcula o endereço do byte (Rs + offset).

(4) Roxo - Memory Access:

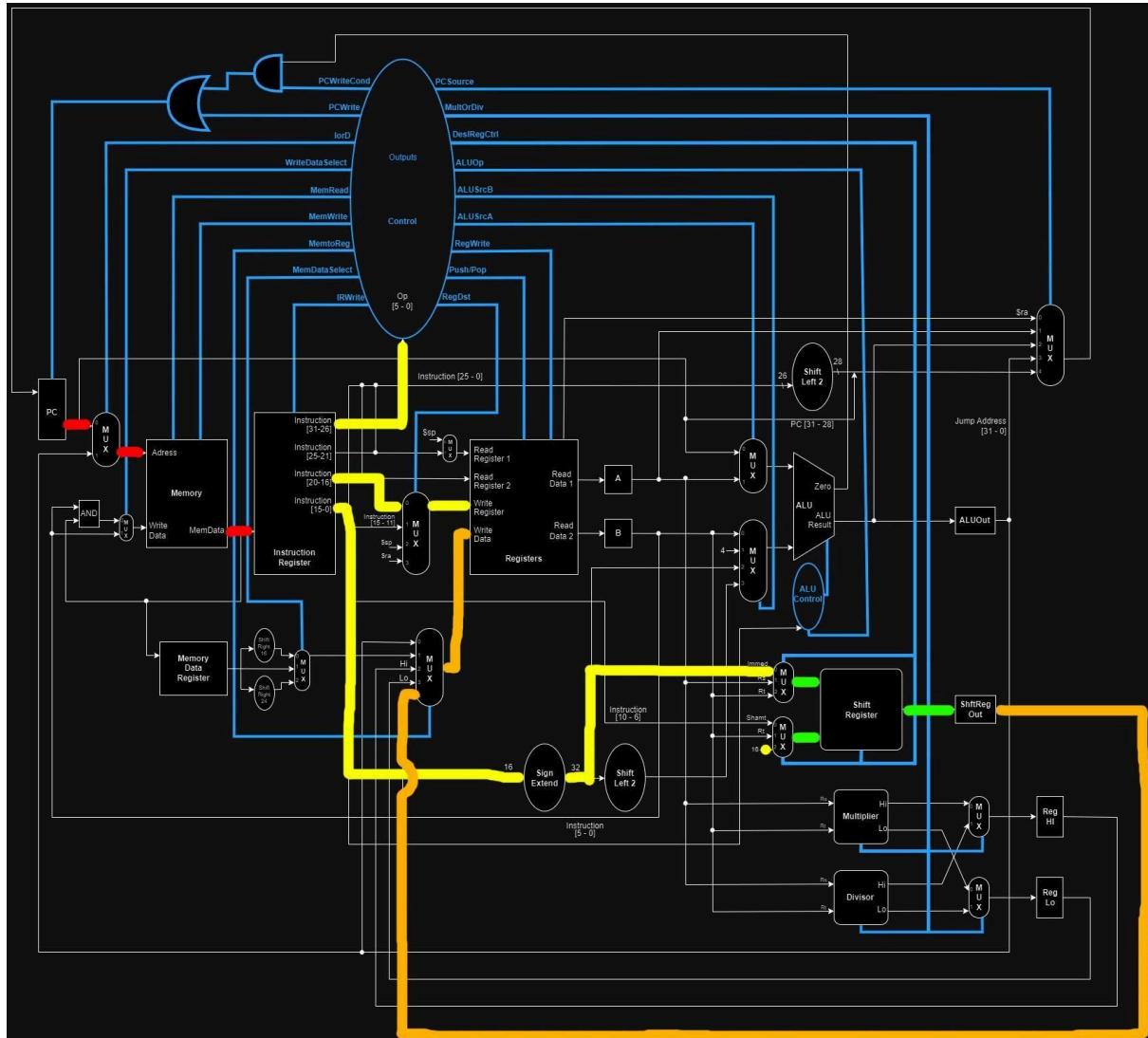
Obtém a palavra (4 bytes) que existe nesse endereço base.

(5) Laranja - Write Back:

Escreve o valor obtido em Rt.

LUI (load upper immediate):

$Rt \leftarrow \text{immediato} \ll 16$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rt e Immediato.

(3) Verde - Execute:

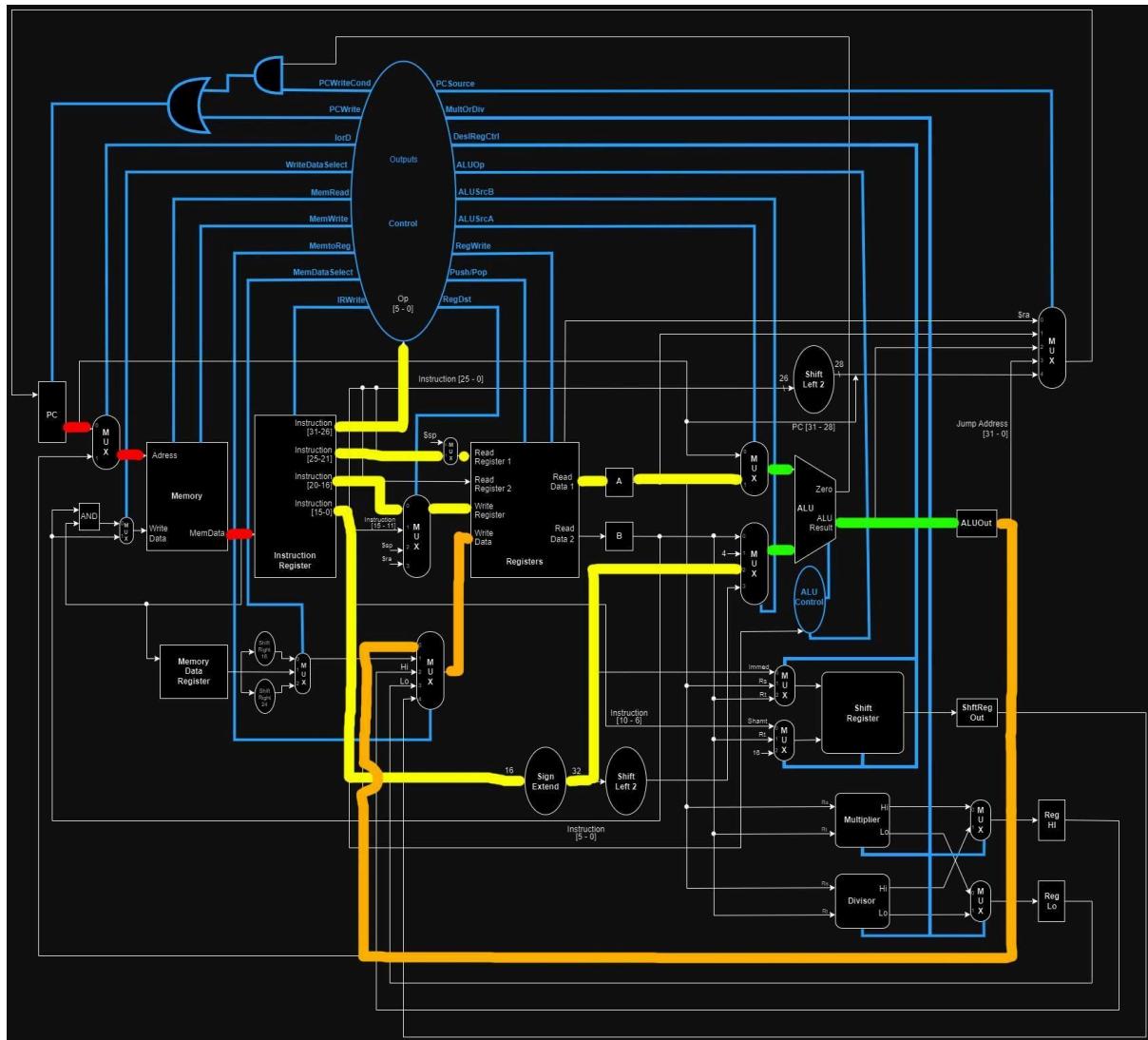
Realiza o shift ($\text{imm} \ll 16$).

(4) Laranja - Write Back:

Escreve o valor obtido em Rt.

SLTI (set less than immediate):

$Rt \leftarrow (rs < imediato) ? 1 : 0$



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e Immediato.

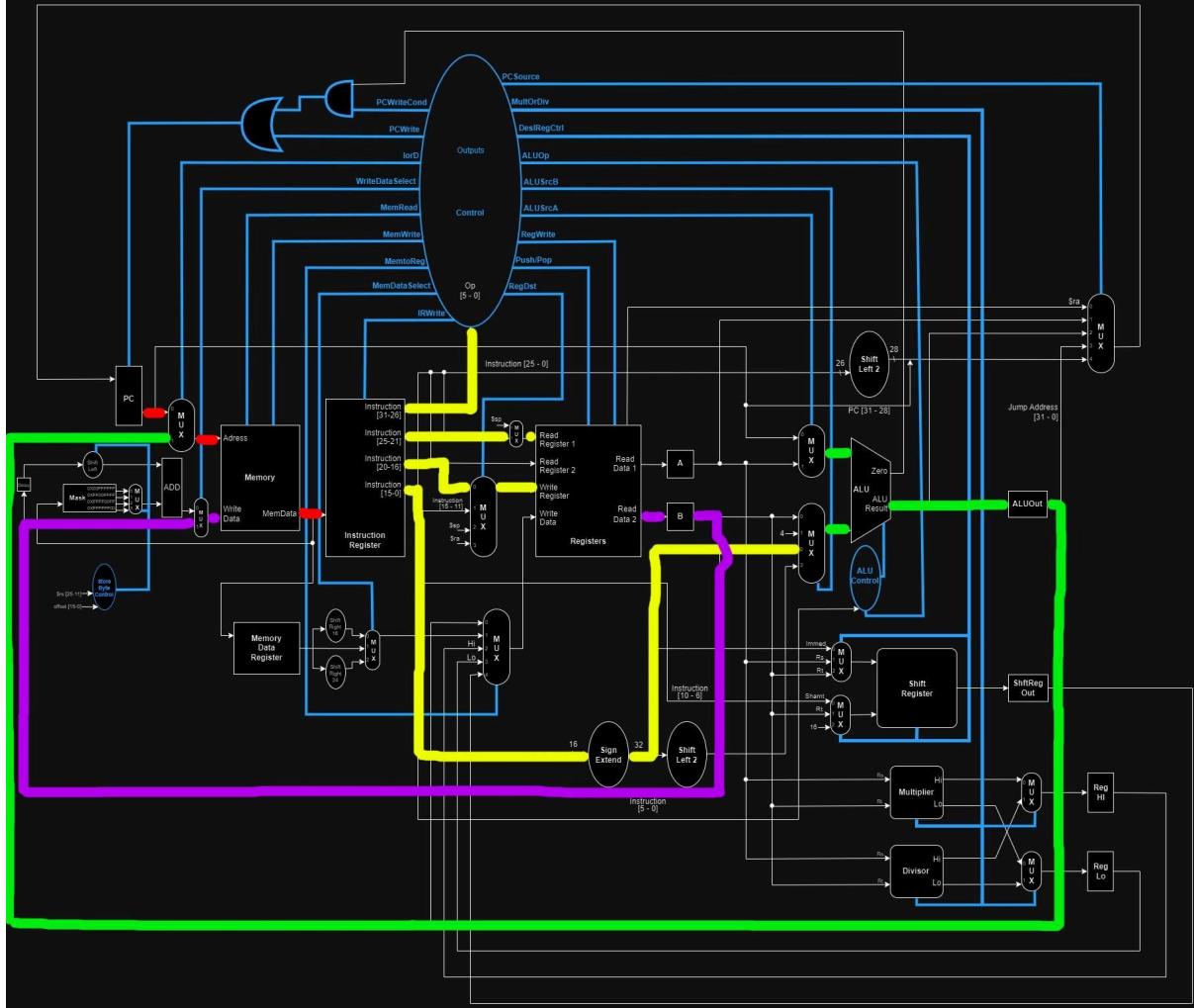
(3) Verde - Execute:

Realiza a comparação ($Rs < Immed$) e armazena o resultado em ALUOut.

(4) Laranja - Write Back:

Escreve o valor obtido em Rt.

SW (store word):
Mem[offset +rs] ← rt



(0) Incremento de PC

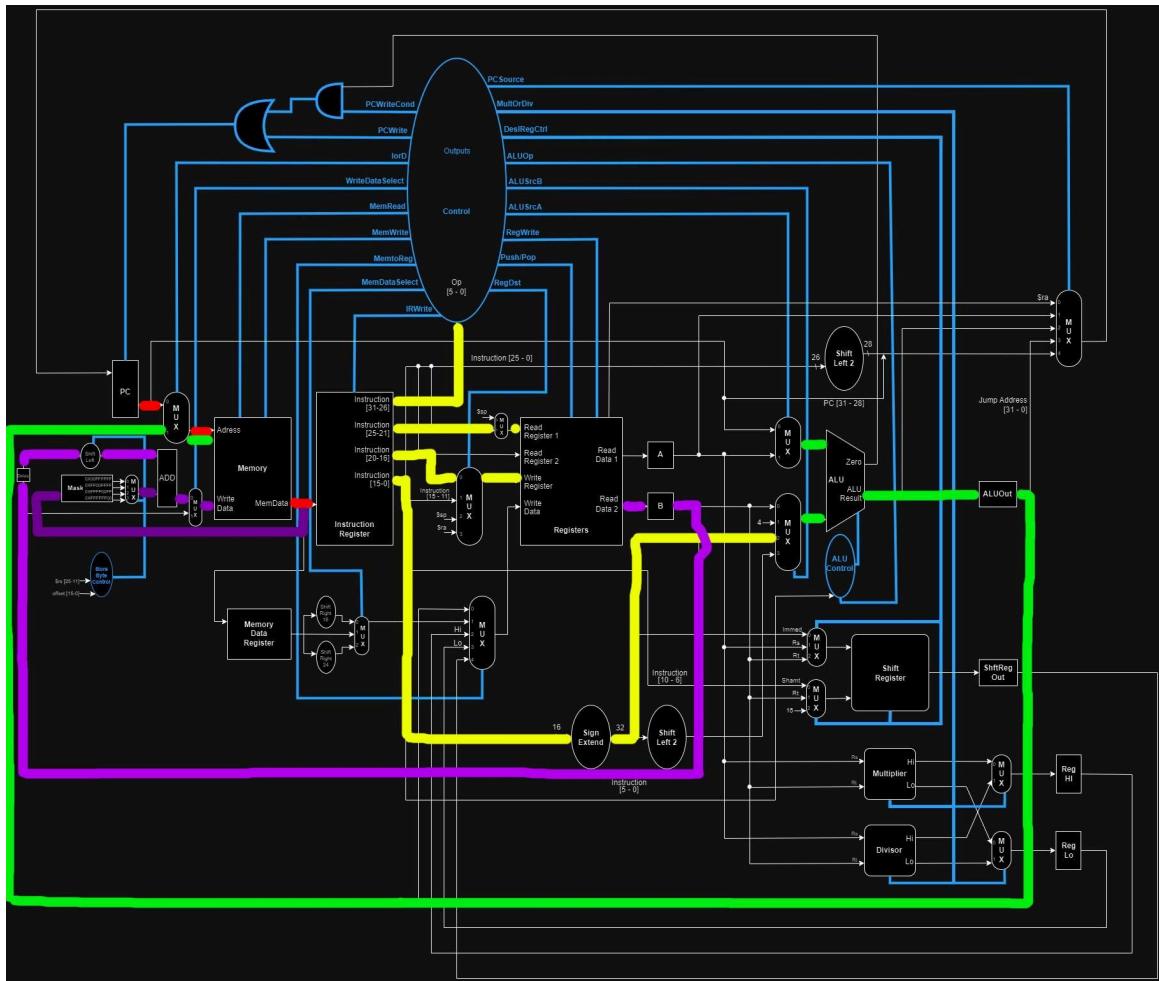
(1) Vermelho - Instruction Fetch:
 Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:
 Lê o opcode e os registradores Rs, Rt e offset.

(3) Verde - Execute:
 Calcula o endereço da palavra (Rs + offset) e encaminha para o MUX de adress.

(4) Roxo - Memory Access:
 Escreve a palavra Rt no endereço calculado.

SB (store byte):



(0) Incremento de PC

(1) Vermelho - Instruction Fetch:

Busca a instrução na memória de instruções.

(2) Amarelo - Instruction Decode:

Lê o opcode e os registradores Rs, Rt e offset.

(3) Verde - Execute:

Calcula o endereço da palavra (Rs + offset) e encaminha para o address da memória.

(4) Roxo - Memory Access:

- (Roxo escuro) = Pega a palavra (P1 P2 P3 P4) armazenada no endereço (rs+offset) e encaminha para uma máscara que zera os bits do byte que será sobrescrito, indicado pelo controle.

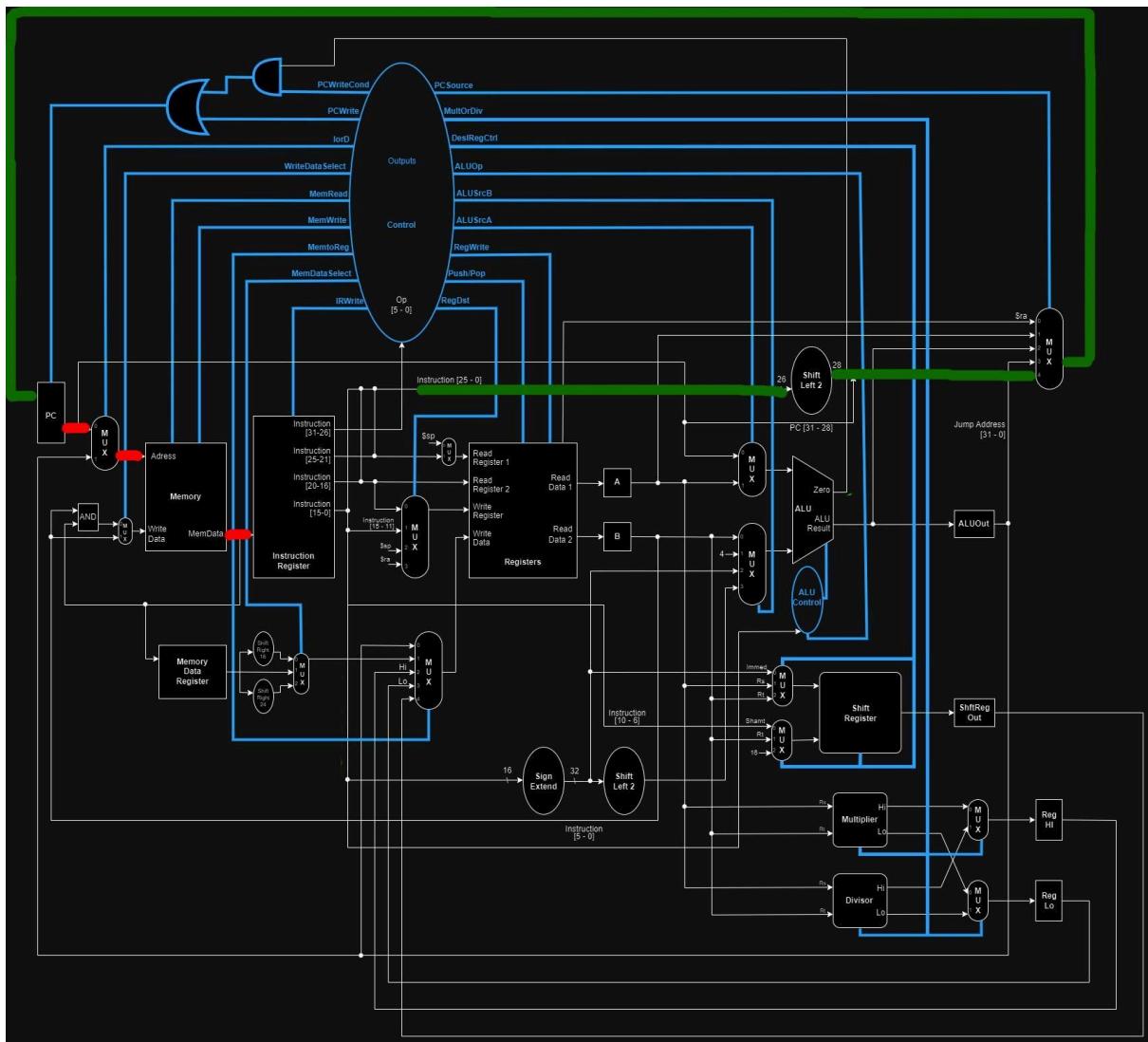
- (Roxo claro) = Encaminha o byte (0 0 0 B), faz (ou não) um sft left a depender da posição (X1 X2 X3 X4) que será inserido, determinado pelo controle.

É feito um ADD entre os dois de modo a juntar a palavra com o byte.

Obs: o “Delay” é utilizado pois o store da palavra demanda mais tempo que o encaminhamento do byte vindo de “B”.

TIPO J

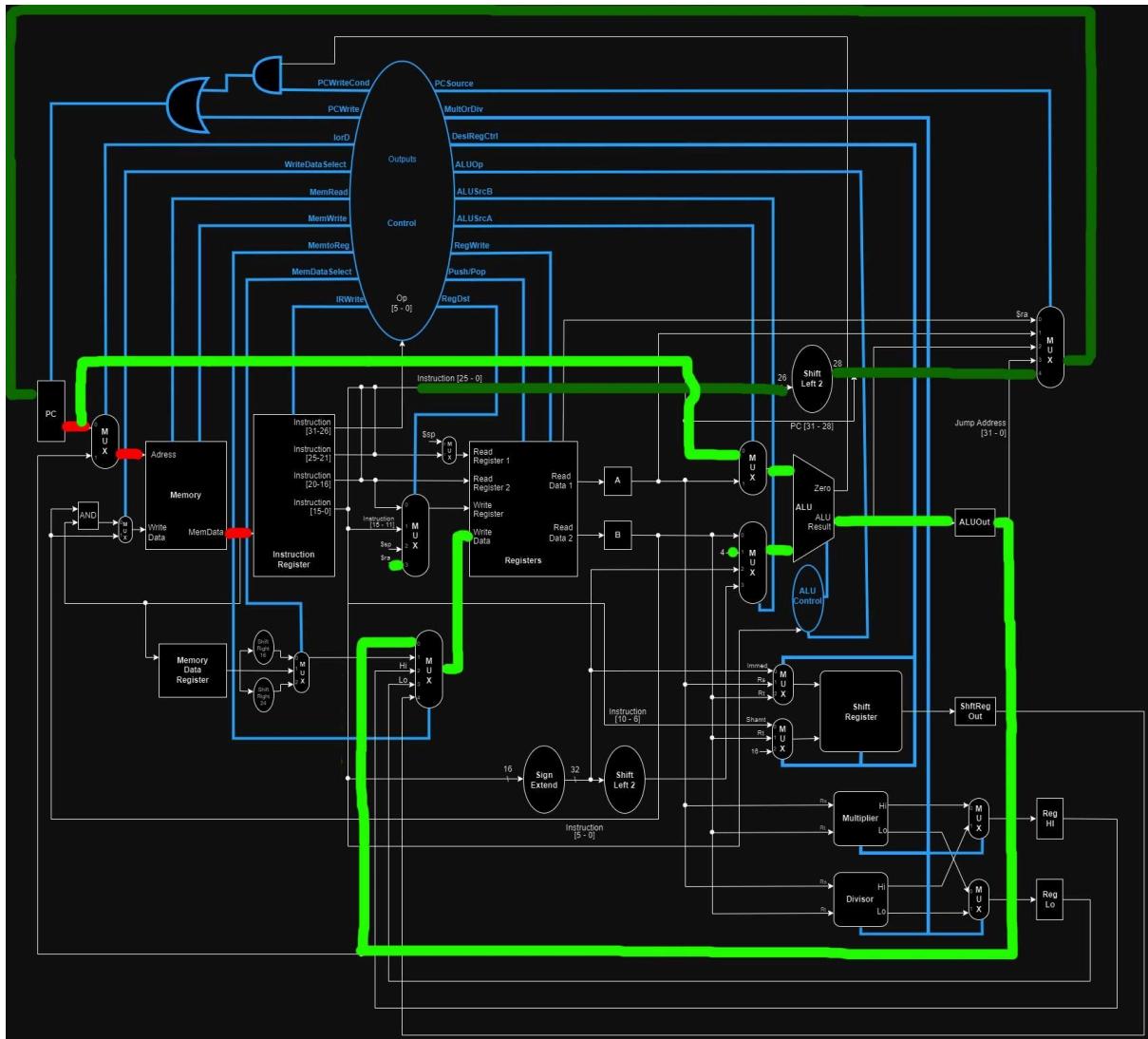
J (jump):



(Vermelho) = Busca a instrução na memória de instruções.

(Verde) = Calcula o endereço de J (offset + upper bits de PC).

JAL (jump and link)



(Vermelho) = Busca a instrução na memória de instruções.

(Verde escuro) = Calcula o endereço de J (offset + upper bits de PC).

(Verde claro) = Armazena o endereço de retorno (PC + 4) em \$ra.