Universidade Federal do Rio Grande do Sul
Instituto de Informática
Departamento de Informática Aplicada
INF01113 - Organização de Computadores B

Relatório da implementação de instruções no MIPS

Alunos:

Eduardo Spitzer Fischer

Gabriel Lando

Rodrigo Paranhos Bastos

Grupo: **4**

Professor: **Luigi Carro**

1. Introdução

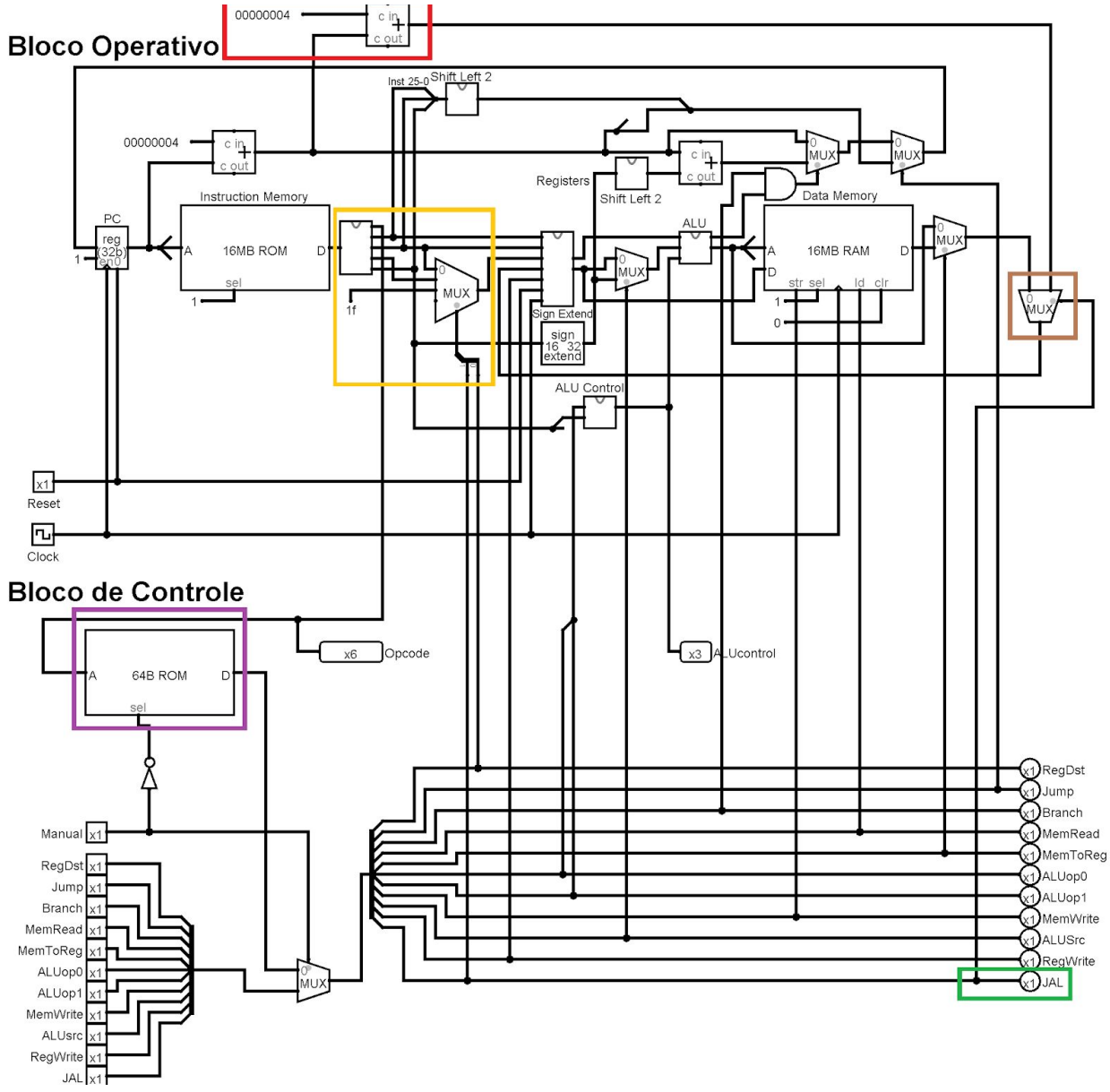
O trabalho consiste na implementação do suporte a 3 novas instruções nos modelos *logisim* das versões do processador MIPS monociclo, multiciclo e com pipeline de 4 estágios. Todos os modelos já possuíam implementadas as instruções *lw*, *sw*, *j* e *beq*. Foram implementadas para cada um dos modelos as instruções *JAL*, *BLEZ* e *SRA*, sem que o funcionamento das originais ficasse prejudicado.

2. Implementação da JAL

JAL -- *Jump and link*

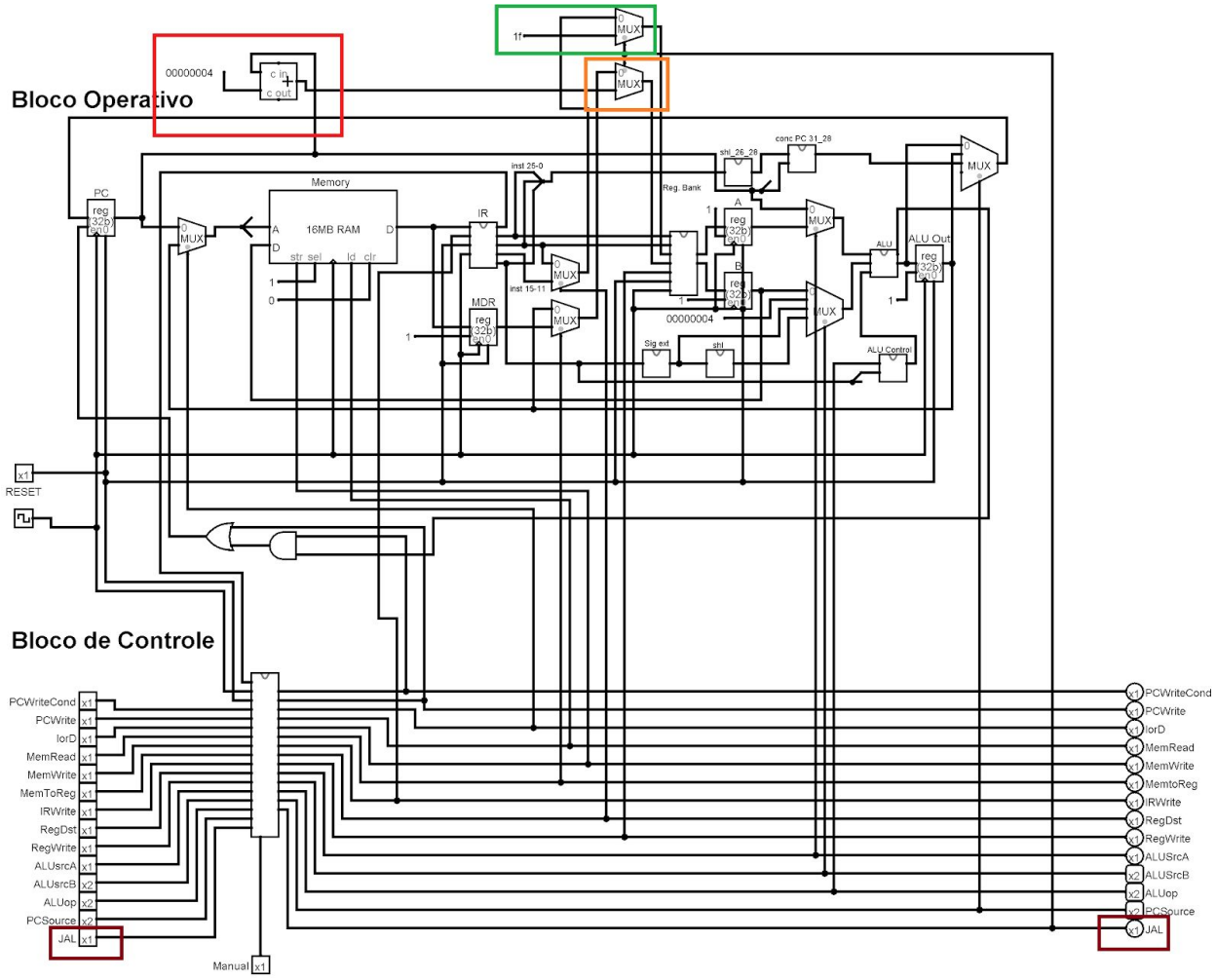
Description:	Jumps to the calculated address and stores the return address in $31
Operation:	$31 = PC + 8 (or nPC + 4); PC = nPC; nPC = (PC & 0xf0000000) | (target << 2);
Syntax:	jal target
Encoding:	0000 11ii iiii iiii iiii iiii iiii iiii

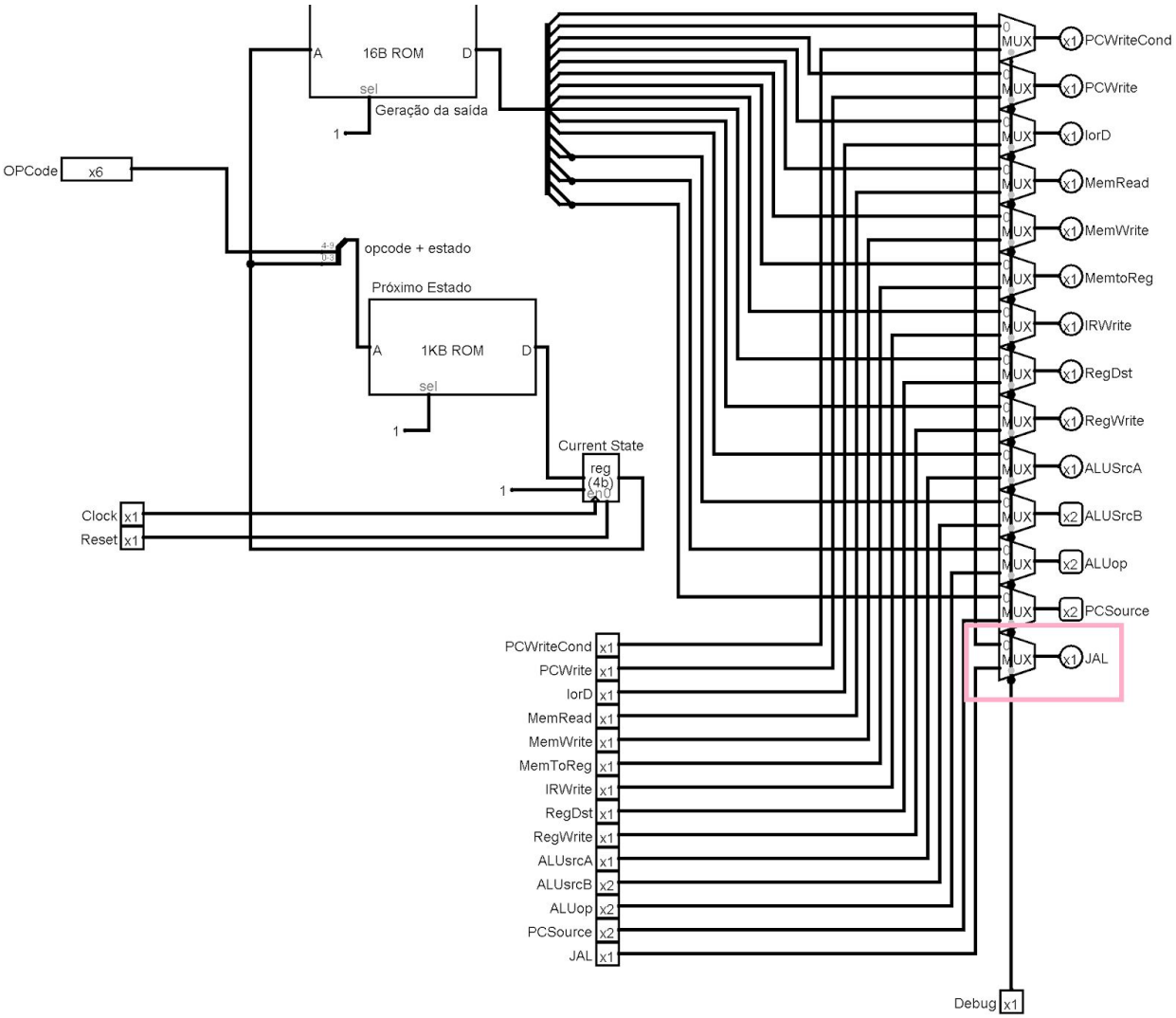
2.1 Alterações no monociclo:



Foi acrescentado ao topo(caixa vermelha) um fio que leva a um somador que adiciona 4 no valor do PC já acrescido de 4, com o objetivo de que esse valor seja guardado no registrador 31 conforme a especificação da instrução. Para que isso ocorra, toda vez que o processador estiver a executar um *jal* este sinal deve ser roteado para a entrada de escrita do banco de registradores. Isso ocorre com o uso de um novo mux, à direita, caixa marrom, que seleciona dentre o que é lido da memória de dados ou que vem da ALU(outro mux determina essa entrada) e o novo sinal. Para o controle deste novo mux foi utilizado um novo sinal de controle, "JAL"(caixa verde), que é ativado com a leitura do opcode da instrução pelo bloco de controle. Para que o bloco de controle(caixa roxa) produzisse os sinais corretos sua memória foi alterada na posição 3, correspondente ao opcode "000011" da instrução *jal* para conter o valor 602, 011000000010 em binário, o que corresponde à ativação dos bits *jump, RegWrite* e *JAL*. Os bits JAL e RegDst controlam o mux modificado(caixa amarela), que agora possui uma entrada constante 0x1f(decimal 31) que é usada para configurar a escrita no registrador 31 quando da execução da JAL.

2.1 Alterações no multiciclo:



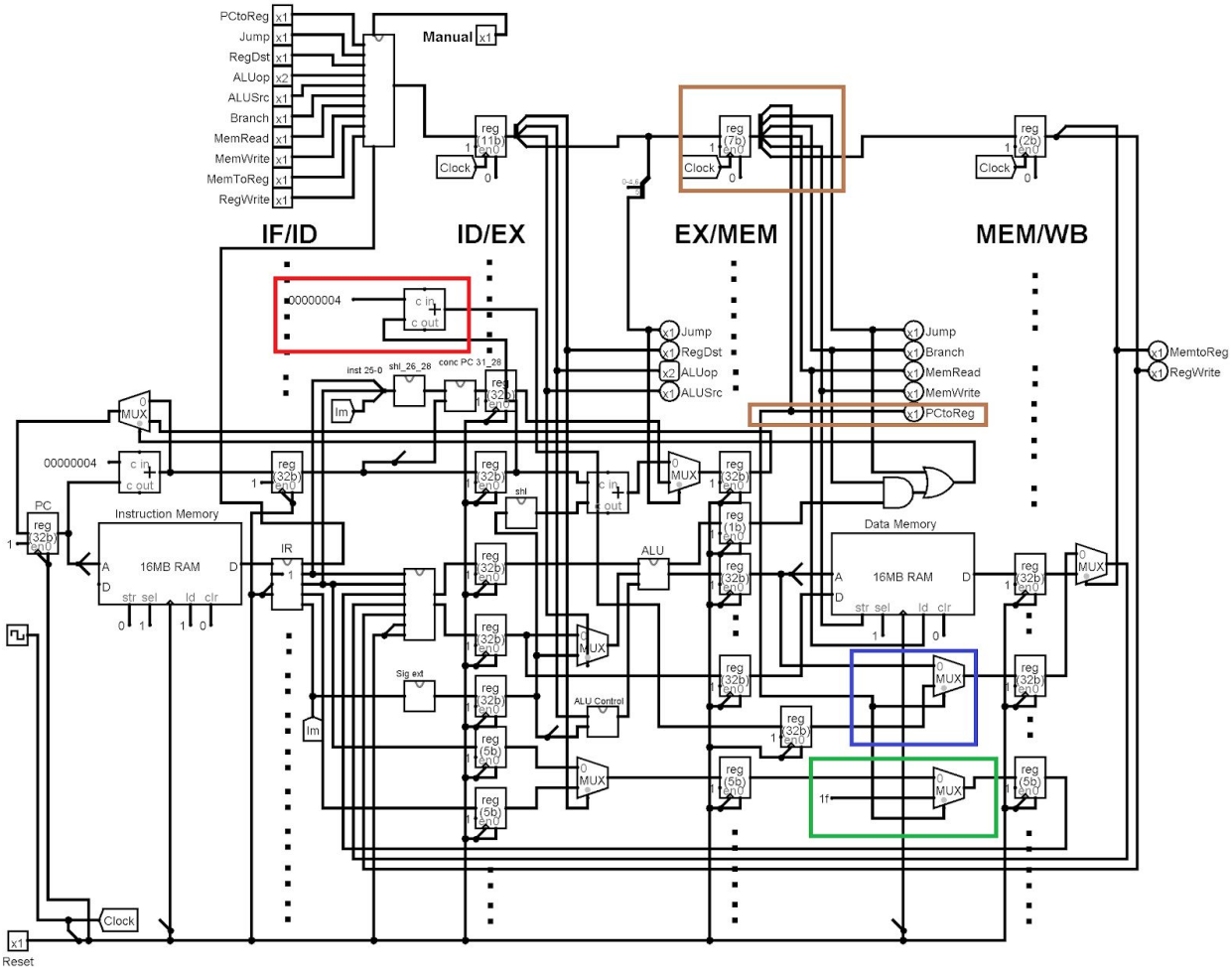

A
16B ROM
D
sel
Geração da saída
1
OPCode
x6
opcode + estado
Próximo Estado
A
1KB ROM
D
sel
1
Current State
reg
(4b)
1
Clock
x1
Reset
x1
MUX
x1
PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
RegDst
RegWrite
ALUSrcA
x2
ALUSrcB
ALUop
PCSource
JAL
PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemToReg
IRWrite
RegDst
RegWrite
ALUsrcA
ALUsrcB
ALUop
PCSource
JAL
Debug

Similarmente ao que foi feito na implementação monociclo foram adicionados um somador e dois multiplexadores. O somador serve ao mesmo propósito de garantir que o endereço de retorno esteja duas posições a frente do endereço em que está a JAL. O multiplexador marcado com verde seleciona o registrador 31(0x1f) para ser escrito quando da execução de JAL. Mais uma vez, é nele que é guardado o endereço de retorno. O multiplexador dentro da caixa laranja verifica se JAL será executada.

Os sinais de controle para cada ciclo ficam:

Ciclo 0: MemRead, ALUSrcA = 0, IorD = 0, IRWrite, ALUSrcB = 01, ALUOp = 00, PCWrite, PCSource = 00.
Ciclo 1: ALUSrcA = 0, ALUSrcB = 11, ALUOp = 00
Ciclo 2: JAL = 1, PCWRITE, RegWrite = 1

2.3 Alterações no pipeline:



As alteraçoes seguem a mesma lógica das das versoes anteriores. Os sinais de controle ficaram:

Ciclo 0: RegDst, ALUop = 10

Ciclo 1: RegWrite, ALUop = 00

Ciclo 2: RegDst, ALUop = 10, RegWrite

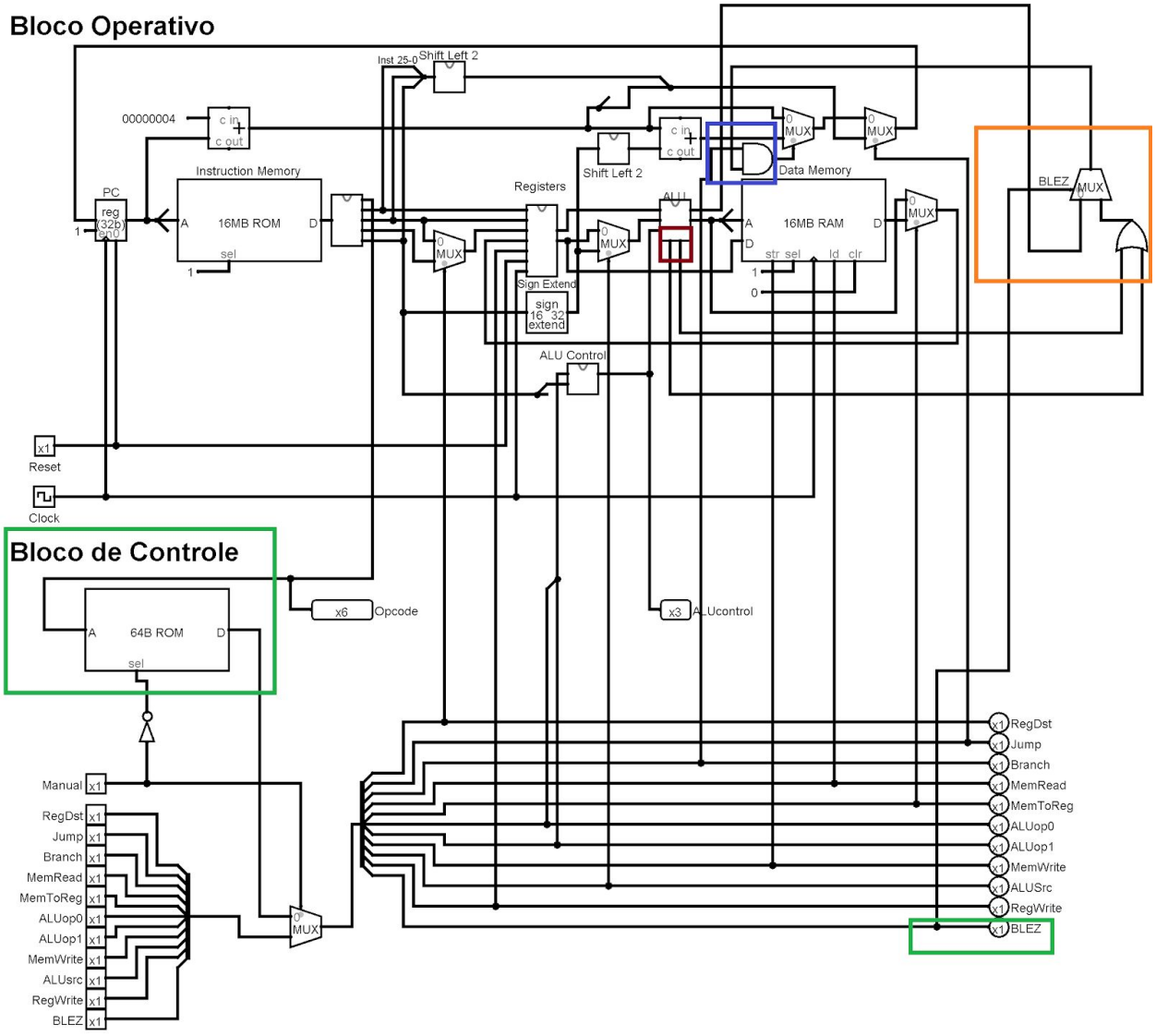
Ciclo 3: RegDst, ALUop = 10

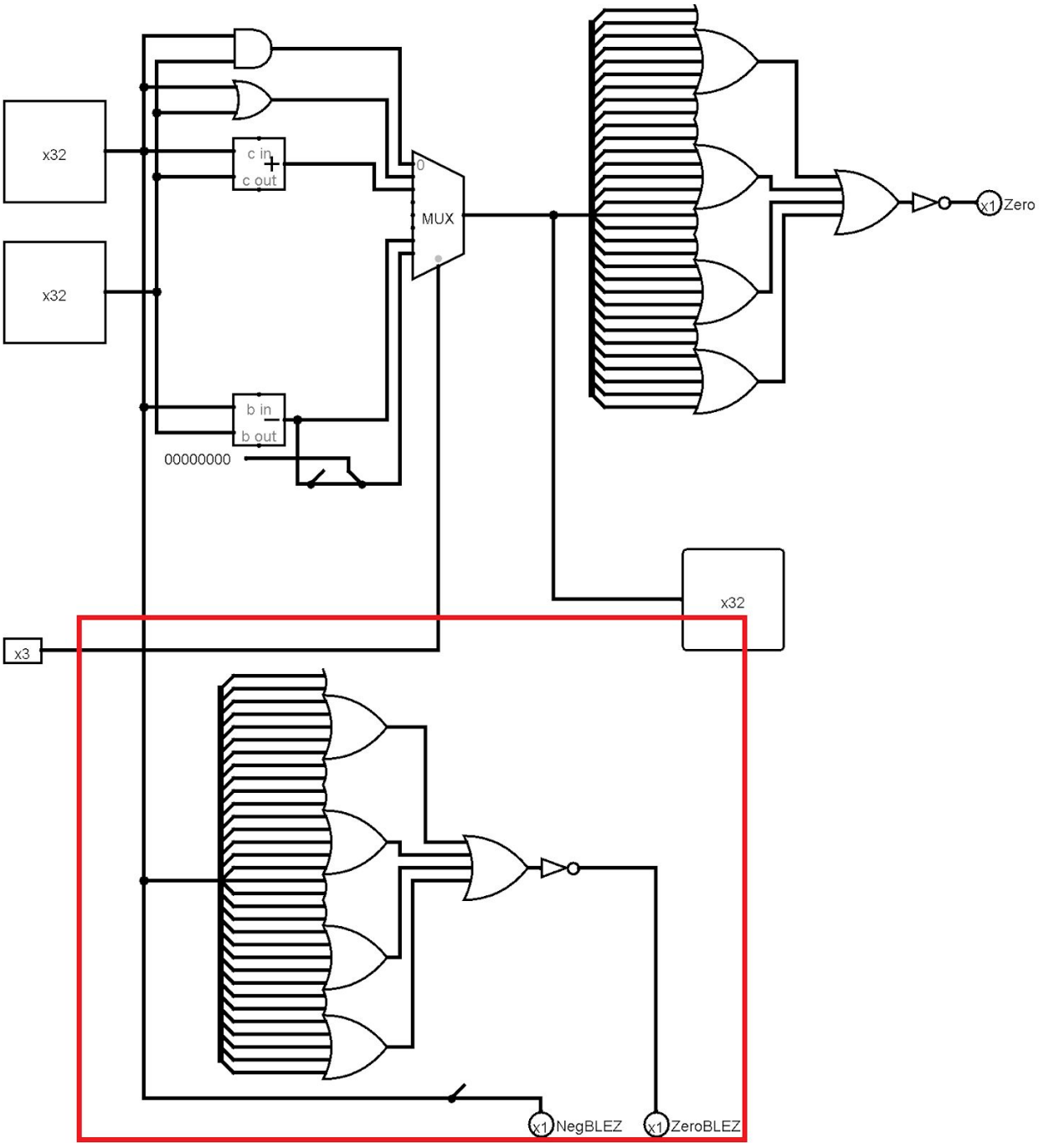
3. Implementação da BLEZ

BLEZ -- ***Branch on less than or equal to zero***

Description:	Branches if the register is less than or equal to zero
Operation:	if $s <= 0 advance_pc (offset << 2)); else advance_pc (4);
Syntax:	blez $s, offset
Encoding:	`0001 10ss sss0 0000 iiii iiii iiii iiii`

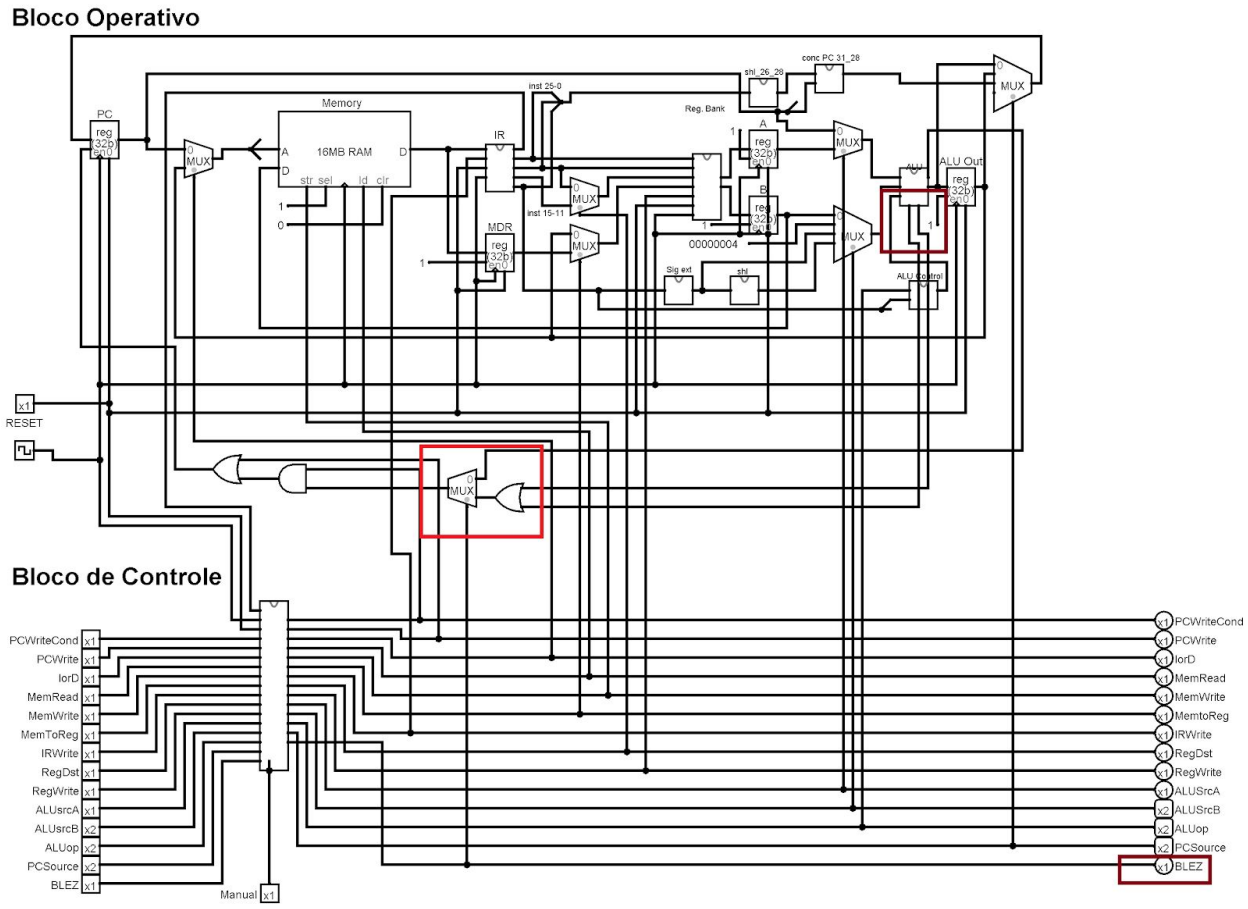
3.1 Alteraçoes no monociclo:




x32
x32
c in
c out
0
MUX
b in
b out
00000000
x1
Zero
x32
x3
x1
NegBLEZ
x1
ZeroBLEZ

Foram adicionados um mux e uma porta OR. Além disso foram criados dois novos sinais saindo da ALU, NegBLEZ e Zero BLEZ. Quando da execução do BLEZ esses novos sinais contornam a lógica da ALU e propagam a informação de se o que o banco de registradores a entrega é negativo ou zero. Esses sinais são ligados a nova porta OR, que está ligada em um novo mux. Esse mux é selecionado pelo sinal de BLEZ, também novo, que é ligado quando da execução da instrução. Todos os outros sinais são idênticos ao de BEQ, de modo que aproveitamos a lógica desta instrução fazendo com que o novo sinal de negativo(negBLEZ) atuasse analogamente ao de zero usado pela BEQ.

3.2 Alterações no multiciclo:



As alterações seguem a mesma lógica da versão monociclo.

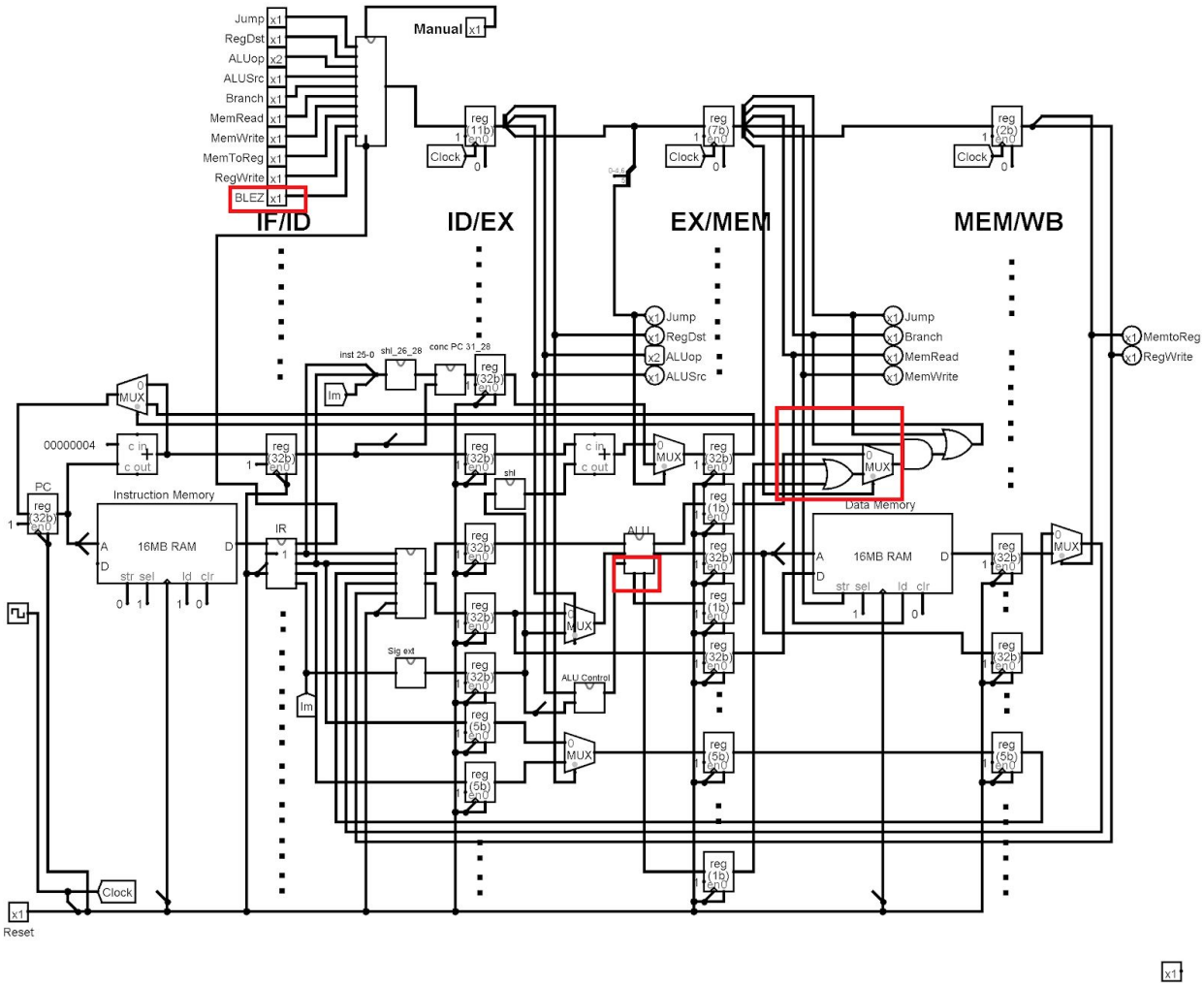
Os sinais de controle ficam:

Ciclo 0: ALUSrcB = 11, ALUop = 00, PCSource = 00, BLEZ = 0

Ciclo 1: PCWriteCond, ALUSrcA, ALUop = 01, PCSource = 01, BLEZ = 1

Ciclo 2: PCWrite, MemRead, IRWrite, ALUSrcB = 01, ALUop = 00, PCSource = 00, BLEZ=0

3.3 Alterações na versão pipeline:



Ciclo 0: RegDst, ALUop = 10, memToReg, RegWrite

Ciclo 1: RegDst, ALUop = 10, RegWrite

Ciclo 2: ALUop = 10, RegWrite

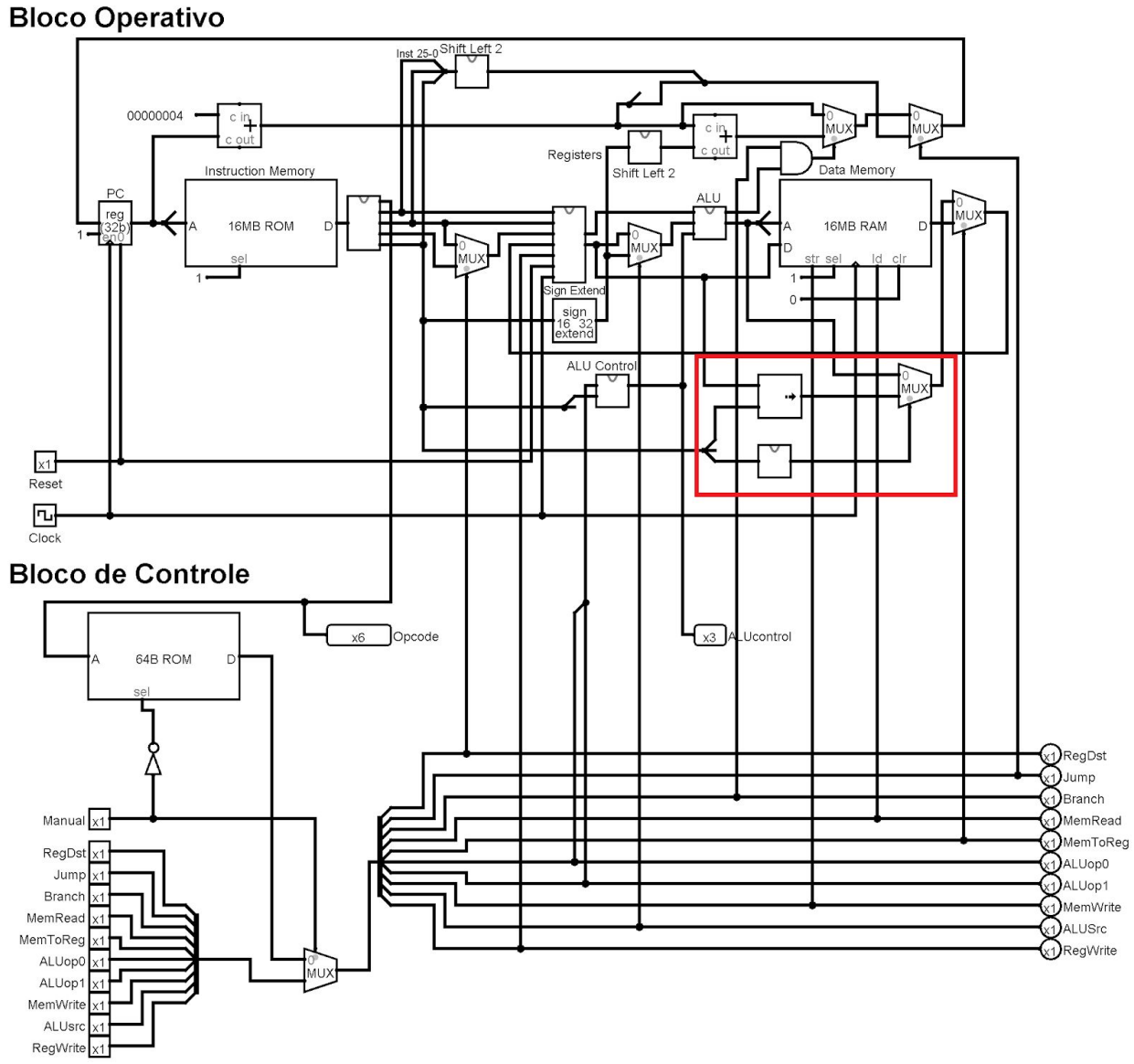
Ciclo 3: RegDst, ALUop = 10, Branch, RegWrite

4. Implementação da SRA

SRA -- ***Shift right arithmetic***

Description:	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. The sign bit is shifted in.
Operation:	$d = $t >> h; advance_pc (4);
Syntax:	sra $d, $t, h
Encoding:	`0000 00-- ---t tttt dddd dhhh hh00 0011`

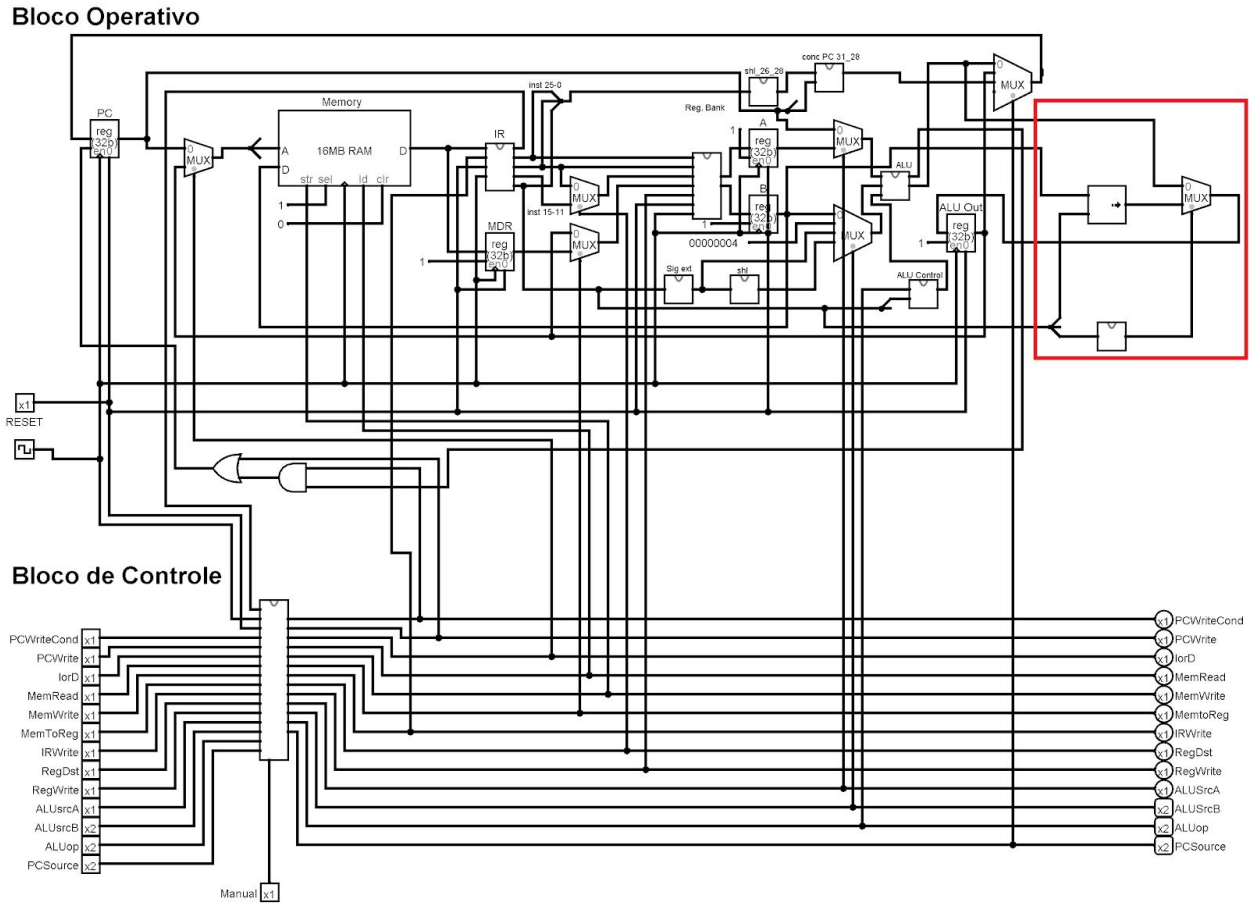
4.1 Alterações no monociclo:

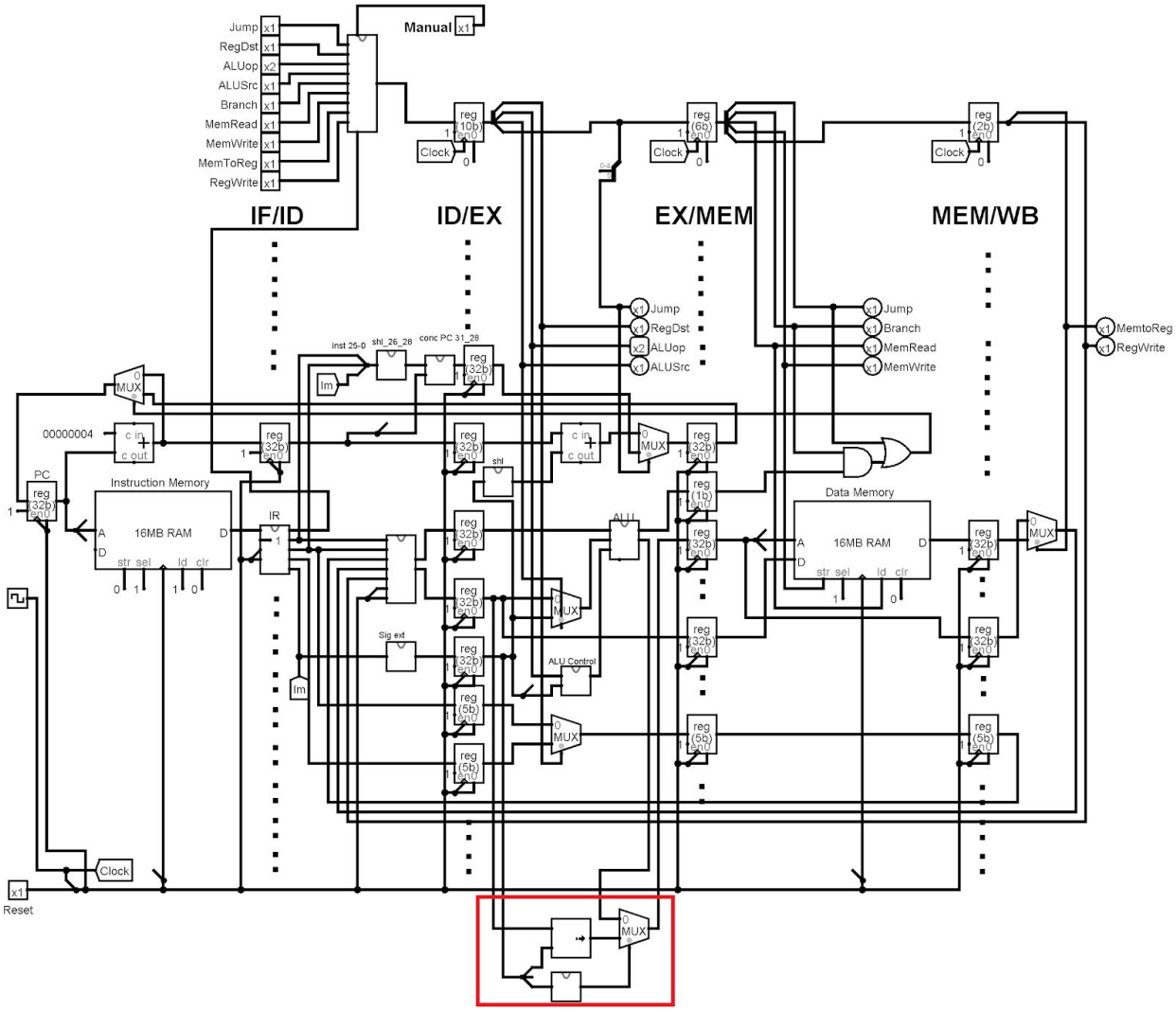


Foram adicionados um comparador e um shifter para a direita, além de um mux para controle. Para o controle deste mux não foi necessário qualquer mudança no bloco de

controle, como era de se esperar visto que o *opcode* da instrução SRA é o mesmo da instrução *add* implementada originalmente. O mux é controlado pelo resultado da comparação do comparador, que compara o *funct* das instruções lidas com o *funct* da instrução SRA(“*hardcoded*”). Quando este vale 1, o mux propaga para o banco de registradores o valor shiftado do segundo registrador, conforme a especificação. A quantidade de bits que devem ser shiftados é extraída do código da instrução com o auxílio de um splitter.

4.1 Alterações no multiciclo e pipeline:





As alterações seguem a mesma lógica das da versão multiciclo. A instrução SRA se mostrou mais simples graças ao fato de que possui o mesmo *opcode* de instruções já implementadas, permitindo que a implementássemos sem mexer na parte de controle exceto pelo uso do comparador e mux já explorados. Os sinais de controle são, portanto, todos os mesmos de outras instruções com o mesmo *opcode*, como a ADD.