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:
Bruno Vivian Carvalho
Carine Bertagnolli Bathaglini
Karine Farias da Silva
Vitor Baldez Maciel
Grupo: 5
Professor: Luigi Carro

Índice

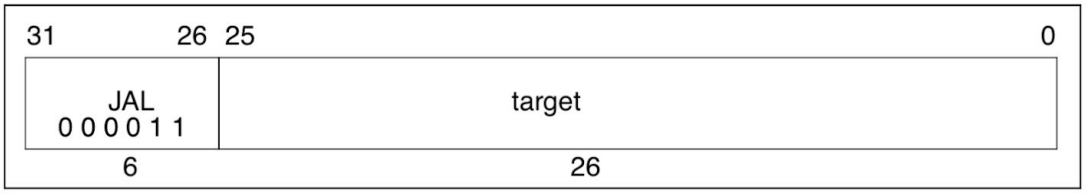
1. Introdução

O trabalho de implementação de instruções nas três versões dos processadores MIPS tem como objetivo nos permitir vivenciar - além do conhecimento teórico nas diferentes organizações - uma abordagem prática.

Para realização do trabalho foi solicitado que o grupo implementasse 4 diferentes instruções nas três versões dos processadores MIPS, de forma que todas as instruções já existentes seguissem em funcionamento. As instruções destinadas ao nosso grupo foram JAL (Jump), BNE (Branch), SRLV (Aritmética/Lógica), LBU (Load) e os resultados serão apresentados nas próximas páginas deste relatório.

2. Implementação da instrução JAL

JAL **Jump And Link** **JAL**



Format:

JAL target

Description:

The 26-bit target address is shifted left two bits and combined with the high-order bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction. The address of the instruction after the delay slot is placed in the link register, *r31*.

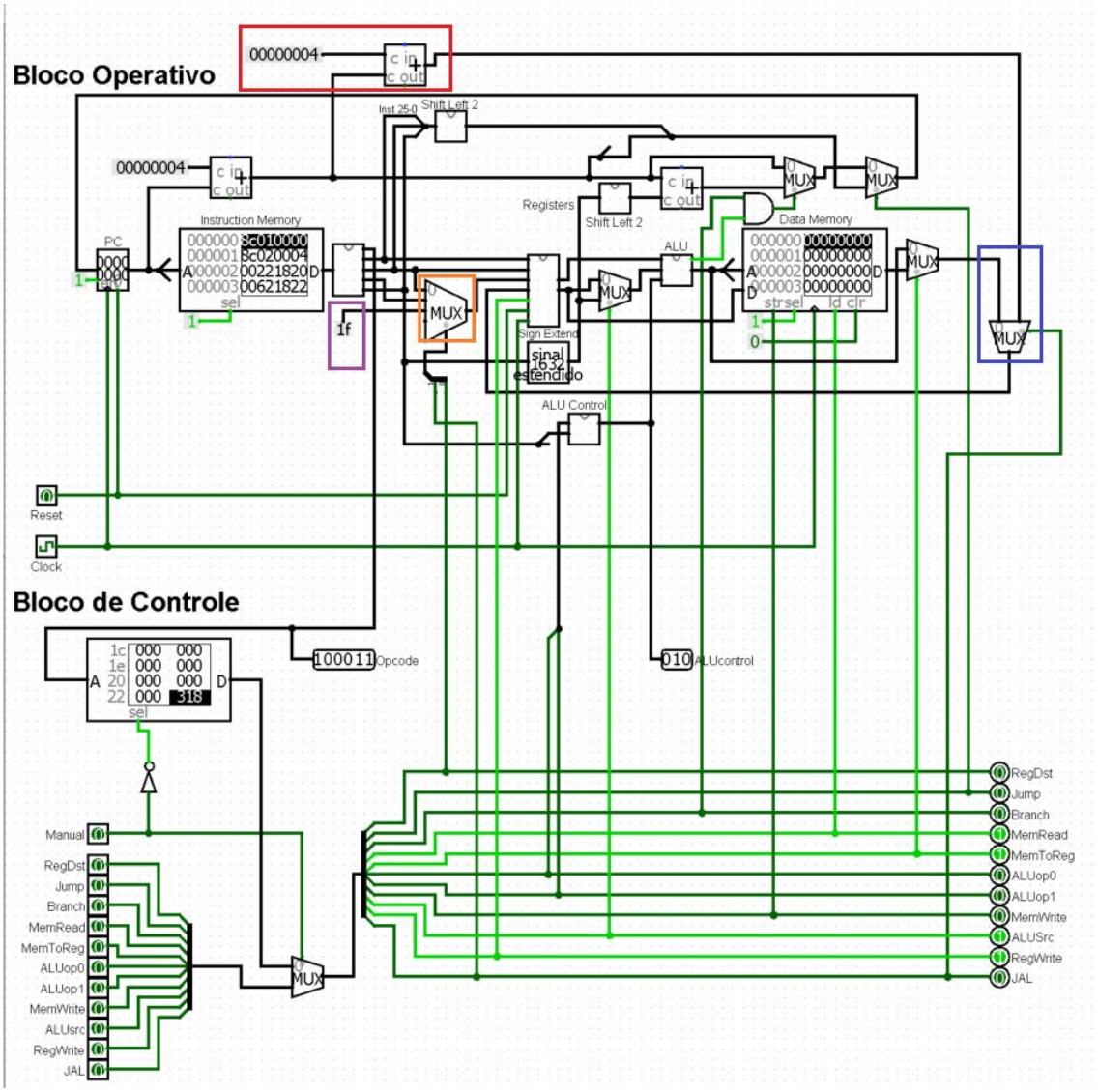
Operation:

```
32    T:    temp ← target
            GPR[31] ← PC + 8
      T+1:  PC ← PC 31...28 || temp || 0^2
64    T:    temp ← target
            GPR[31] ← PC + 8
      T+1:  PC ← PC 63...28 || temp || 0^2
```

Exceptions:

None

2.1 - Alterações no monociclo:



Foram acrescentados um somador com +4 (em vermelho) pois o programa salta incondicionalmente para o endereço calculado com um atraso de uma instrução. O multiplexador (em laranja) foi alterado para ter a entrada do registrador 31(1f, em roxo) que será armazenado o endereço seguinte a instrução JAL. E o último mux (em azul) foi adicionado para definir se é a instrução JAL que será executada. Os pinos de entrada e saída da JAL também foram acrescentados.

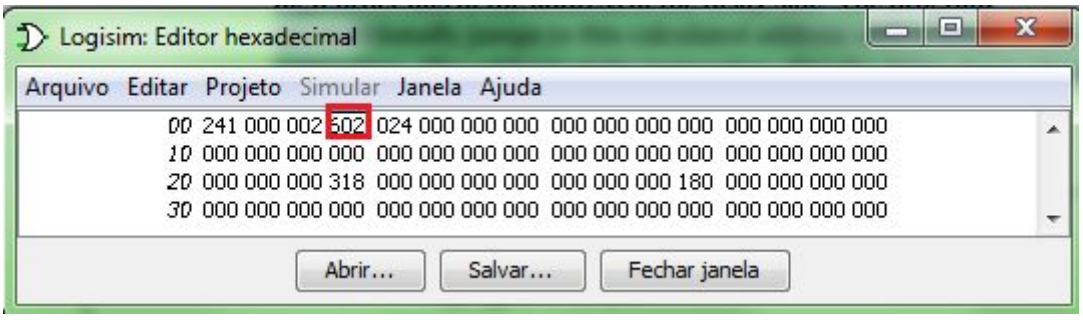
Novas instruções no MIPS foram acrescentadas:

00. INÍCIO: LW R1, 0(R0)
01. LW R2, 4(R0)
02. ADD R3, R1, R2
03. SUB R3, R3, R2
04. BEQ R3, R1 LB1
05. SW R0, 8(R0)
06. LB1: SW R3, 8(R0)
07. JAL FUNC
08. LW R3,0(R0)
09.J 0
10. FUNC: SW R3, 8(R0)
11. JR #31 (não implementado, por ser uma instrução diferente da pedida)

A instrução JAL, do tipo J, que foi acrescentada na memória de instruções foi 0C00000Ahex, quando convertida para binário ficou:

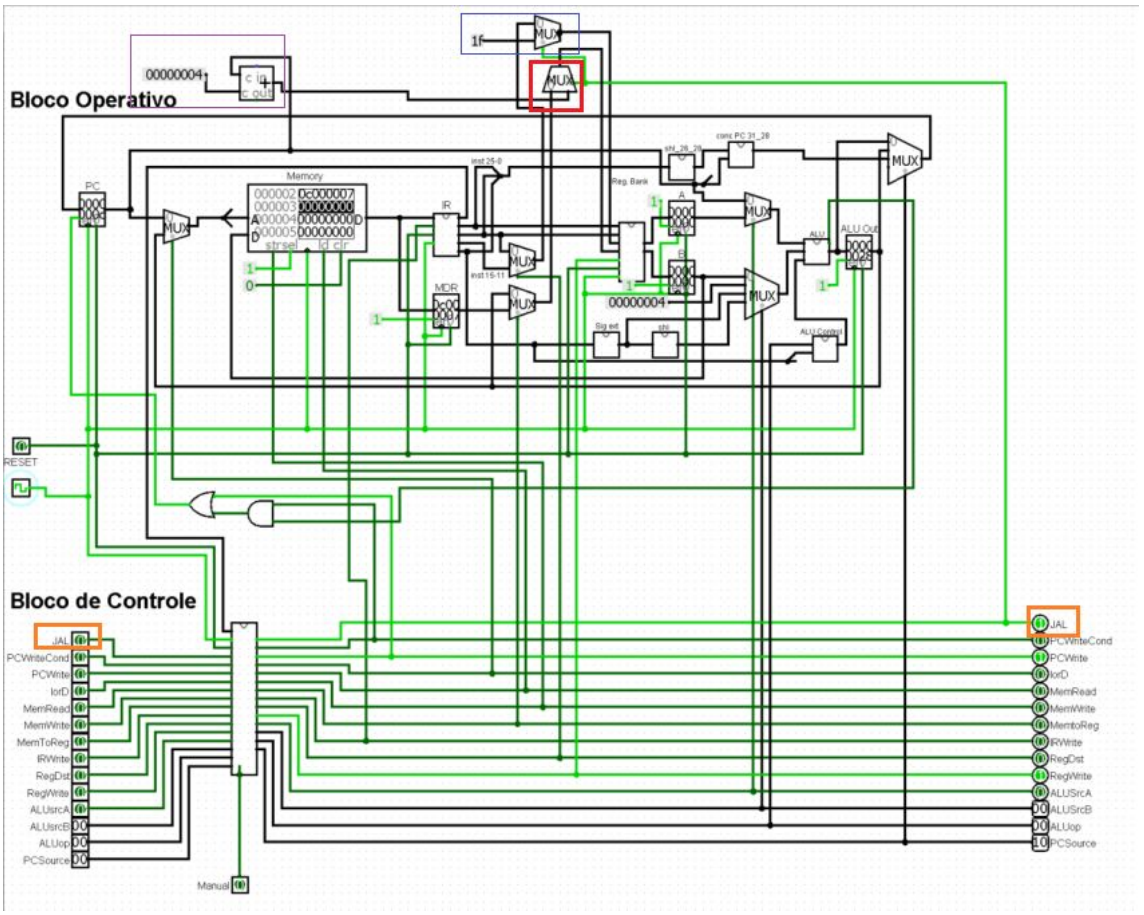
000011	00000000000000000000001010
JAL	endereço de destino
op (6 bits)	endereço (26 bits)

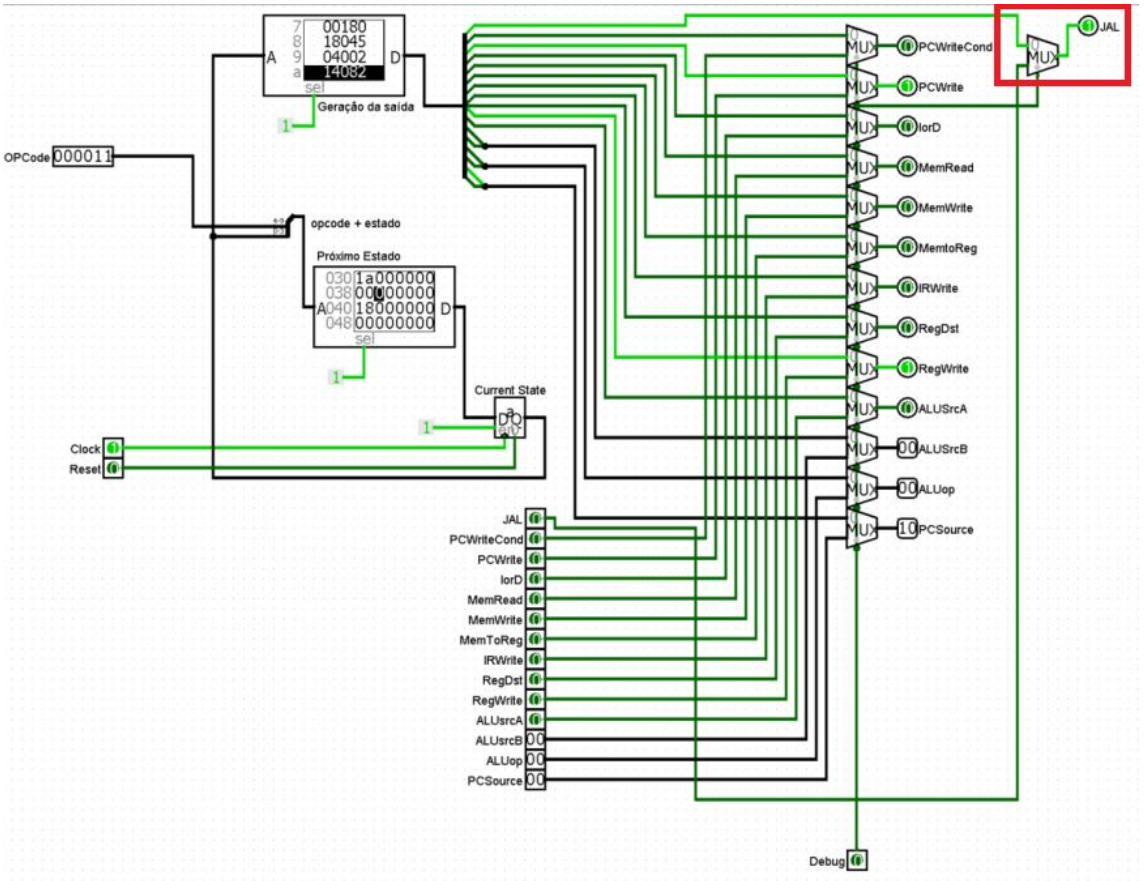
Outro mudança necessária foi adicionada ao bloco de controle:



Onde 602 representa o valor 011000000010 em binário. Esses bits correspondem a ativação do Jump, RegWrite e JAL selecionados.

2.2 - Alterações no multiciclo:





Foram acrescentados dois multiplexadores e um somador com +4. O multiplexador (em azul) seleciona o registrador 31 caso a instrução JAL esteja ativa para salvar o endereço próximo a instrução JAL neste registrador. O multiplexador (em vermelho) foi adicionado para definir se é a instrução JAL que será executada. O somador com +4 (em vermelho) foi acrescentado pois o programa salta incondicionalmente para o endereço calculado com um atraso de uma instrução.

No bloco de controle foi colocado o número 14082 em hexadecimal que corresponde a 0001010000010000010 em binário. Cada bit ativado neste número corresponderá a um fio ligado aos pinos de saída.

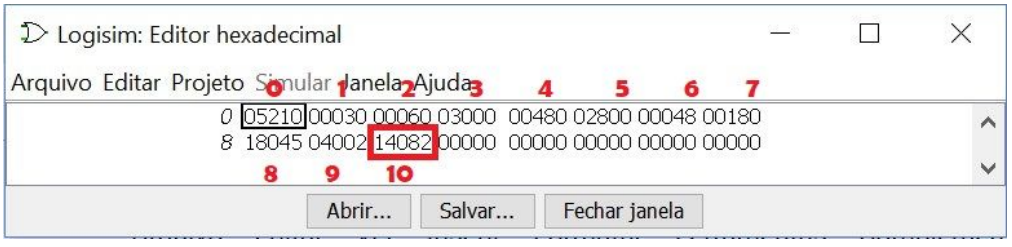
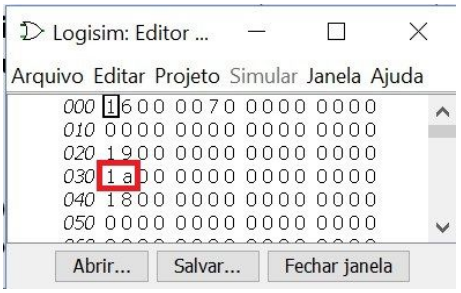
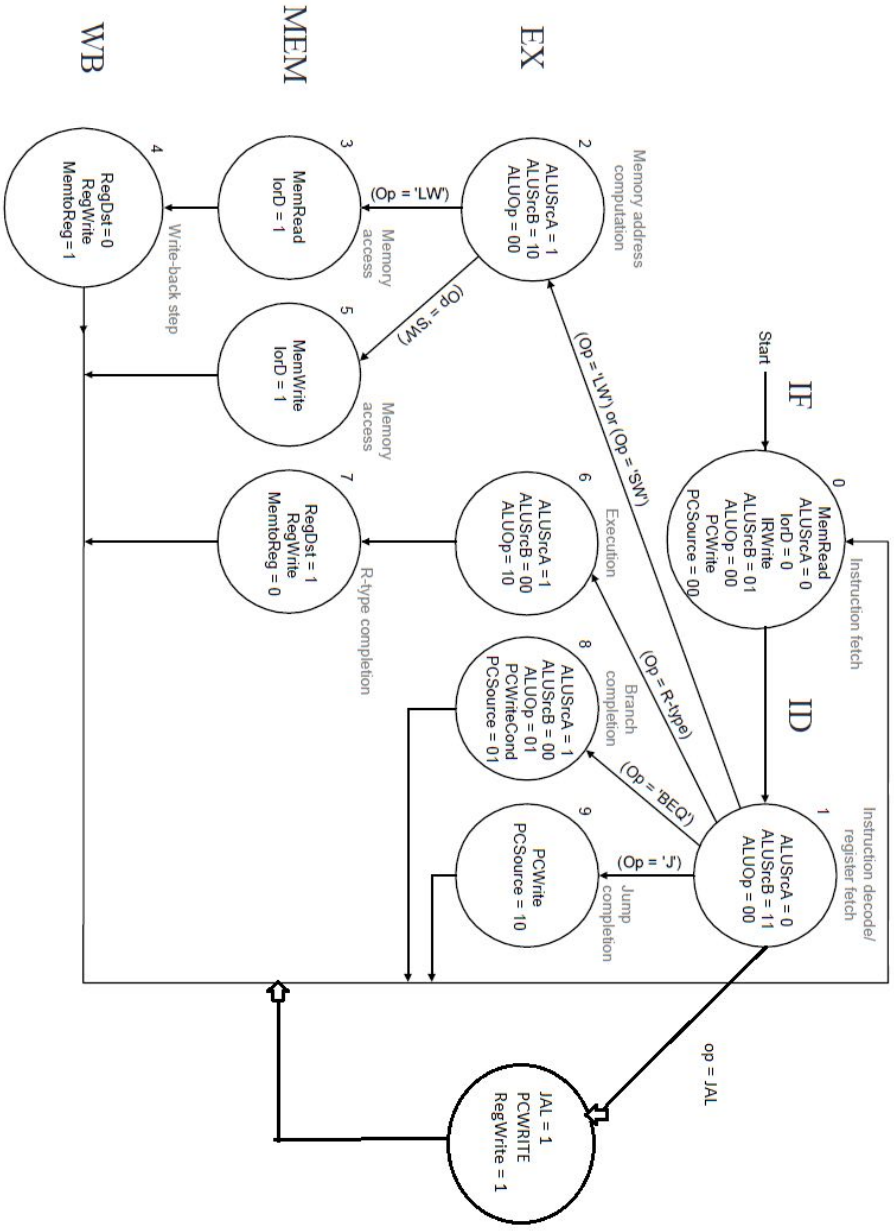


Figura 01: ROM no bloco de controle

Outra alteração no bloco de controle foi a inserção do 1 a na linha 030. A tradução para essa linha é que a instrução JAL irá executar, a cada ciclo do clock, o que está no bloco 1 (figura 01) primeiro e depois executará o que está no bloco 10 da mesma figura.

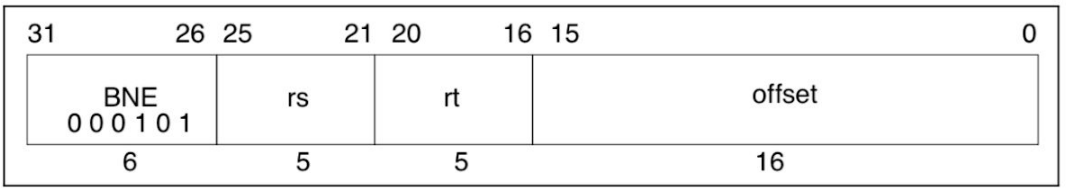


O diagrama de estados ficou da seguinte forma:



Implementação da instrução BNE

BNE **Branch On Not Equal** **BNE**



Format:

BNE rs, rt, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are not equal, then the program branches to the target address, with a delay of one instruction.

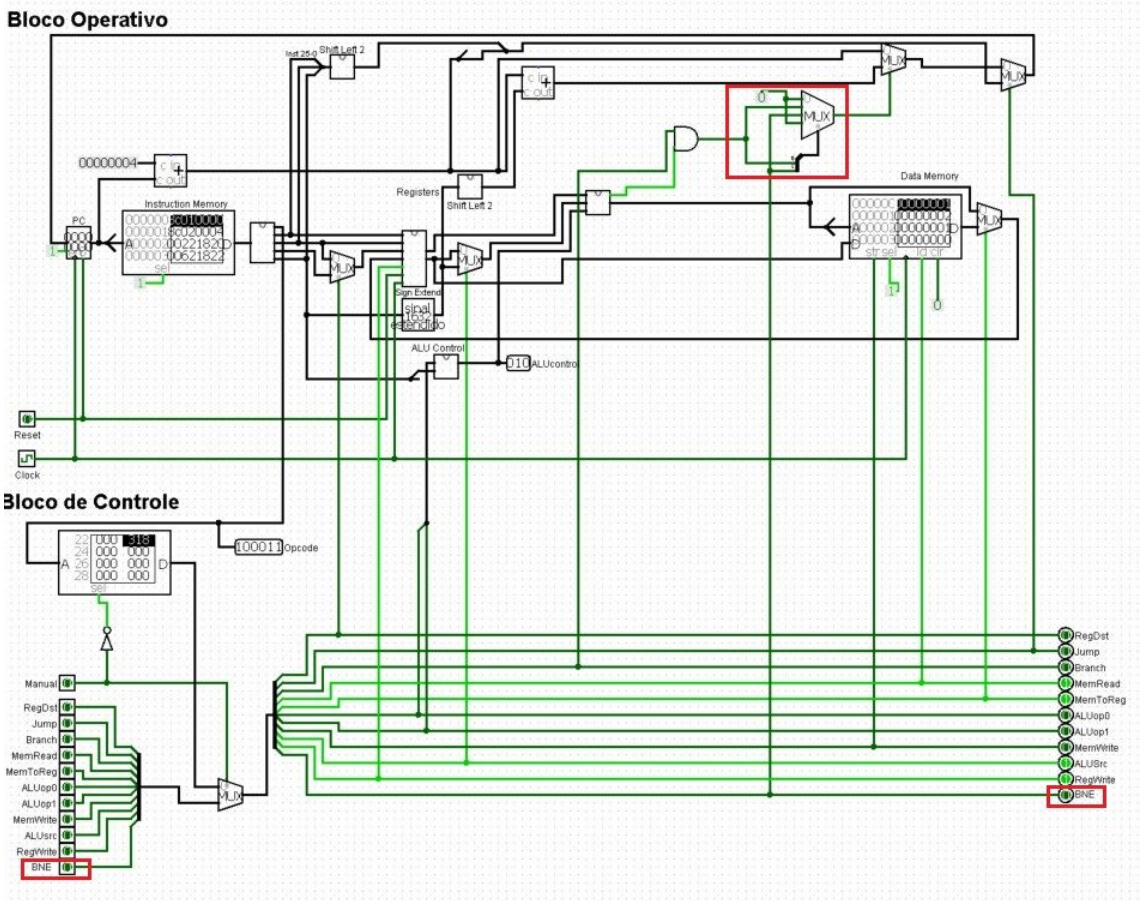
Operation:

```
32   T:    target ← (offset15)^14 || offset || 0^2
           condition ← (GPR[rs] ≠ GPR[rt])
      T+1: if condition then
                PC ← PC + target
           endif
64   T:    target ← (offset15)^46 || offset || 0^2
           condition ← (GPR[rs] ≠ GPR[rt])
      T+1: if condition then
                PC ← PC + target
           endif
```

Exceptions:

None

3.1 - Alterações no monociclo:



Foram acrescentados um multiplexador com dois bits para a seleção, um distribuidor para facilitar a seleção no mesmo, bem como um pino de entrada e outro de saída que correspondem ao BNE.

O primeiro passo foi acrescentar novas instruções no MIPS: (exemplo de finalização do código)

```
00. INÍCIO: LW R1, 0(R0)          # R1 = memóriaDeDados[R0 + 0]
01. LW R2, 4(R0)                   # R2 = memóriaDeDados[R0 + 4]
02. ADD R3, R1, R2                 # R3 = R1 + R2
03. SUB R3, R3, R2                 # R3 = R3 - R2
04. BEQ R3, R1 LB1                 # if(R3 == R1) then LB1
05. SW R0, 8(R0)                   # memóriaDeDados[R0 + 8] = R3
06. LB1: SW R3, 8(R0)              # LB1: memóriaDeDados[R0 + 8] = R3
07. BNE R3,R1,NOT_EQUAL            # if(R3 != R1) then NOT_EQUAL
08. j 0                            # j INÍCIO
09. NOT_EQUAL: SW R3, 20(R0)       # memóriaDeDados[R0 + 20] = R3
10. J 0                            # j INÍCIO
```

A instrução BNE, do tipo I, acrescentada na memória de instruções, que representa a linha 07, foi 14410001hex e que quando convertida para binário:

000101	00010	00001	0000000000000001
BNE	R2**	R1**	1*
op (6 bits)	rs (5 bits)	rt (5 bits)	endereço (16 bits)

*O número 1 é explicado pelo fato de que, caso os registradores (R1 e R2) sejam diferentes só será pulada uma instrução (j na linha 08). Caso fossem puladas mais instruções a quantidade saltada preencheria o campo do endereço na tabela.
**Quaisquer outros registradores poderiam ter sido usados para exemplo de comparação.

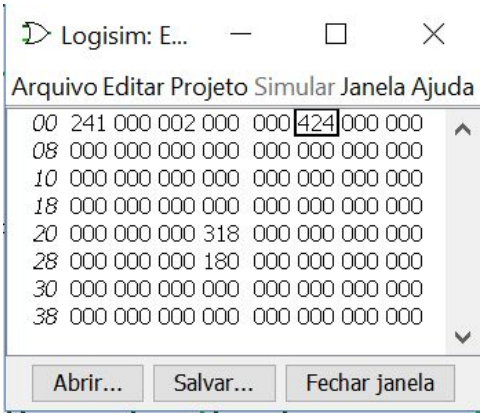
A utilização do multiplexador se deu em vista de diferenciar os branches - "*branch-equal*" e "*branch-not-equal*". A seleção desse mux é feita pelo pino de entrada no BNE e pela resposta da instrução BEQ (que veio implementada no MIPS).

BNE	Branch		
0	0	0	não tem ocorrência de nenhum tipo de branch
0	1	1	apenas a instrução branch
1	0	1	apenas a instrução BNE
1	1	0	

Outra mudança significativa é a inclusão no bloco de controle do número:

424 hex = 010000100100 binário.

Esses bits corresponderão as linhas do BNE, Branch e ALUop0 selecionadas.



3.2 - Alterações no multiciclo:

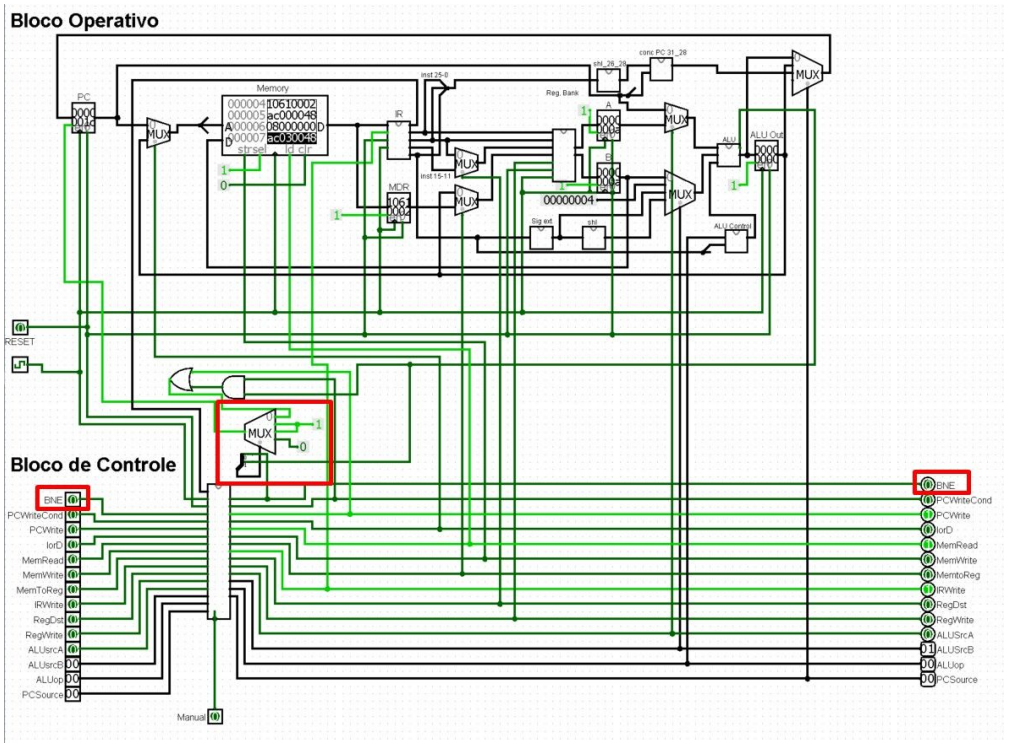


Figura 01: MIPS Multiciclo com a instrução BNE implementada.

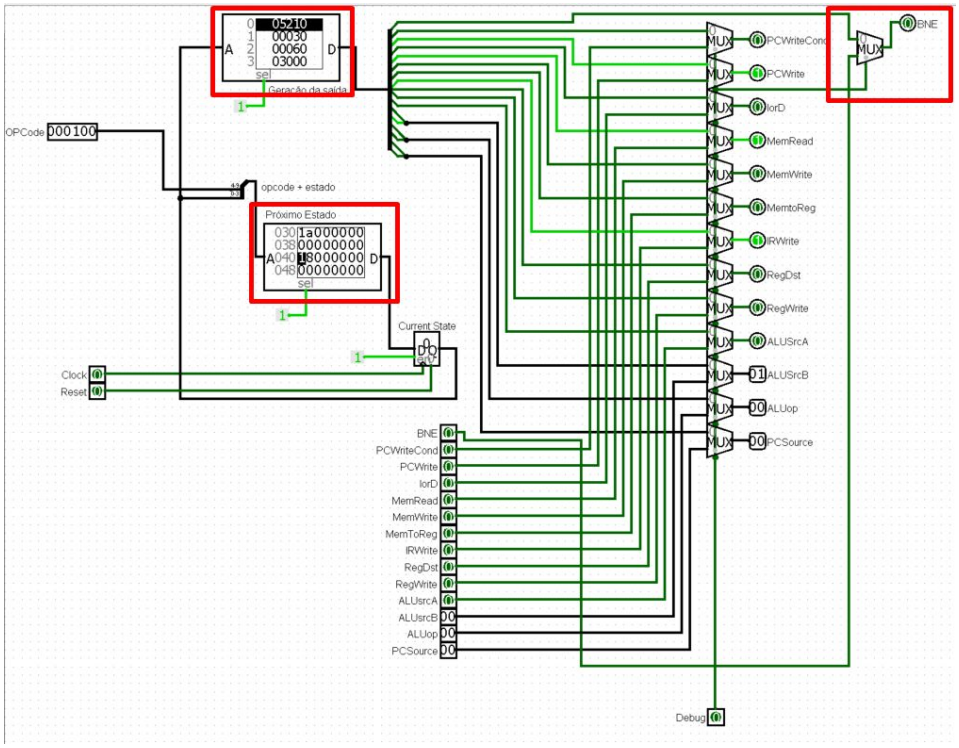
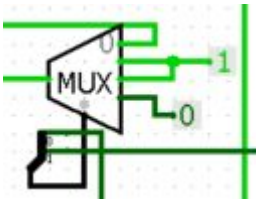


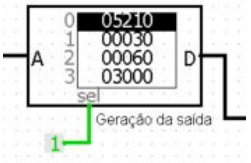
Figura 02: MIPS Multiciclo - bloco de controle

Foram acrescentados um multiplexador com 2 bits para seleção e com 1 bit para entrada de dados, um pino de entrada e um pino de saída. Sabendo que se dois registradores são iguais ocorre o ativamento do "zero" na ULA a lógica foi a seguinte:



BNE	zero da ULA = 1	saída
0	0	valor da or que já estava no circuito
0	1	1
1	0	1
1	1	0

No bloco de controle foi colocado o número 18045 em hexadecimal que corresponde a 0001100000001000101 em binário. Cada bit ativado neste número corresponderá a um fio ligado aos pinos de saída.



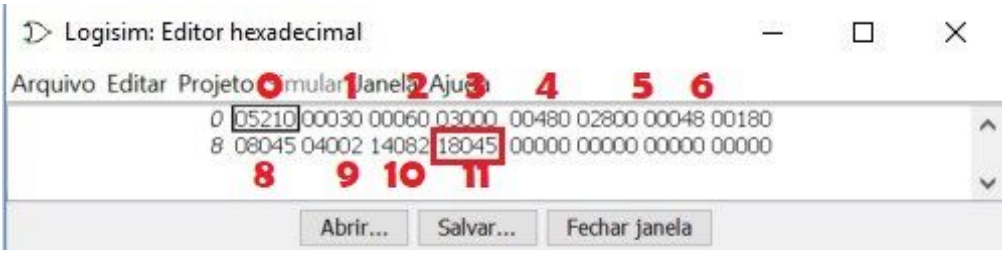


Figura 03: ROM no bloco de controle, controlará quais fios de saída serão ligados.

Outra alteração no bloco de controle foi a inserção do 1 b na linha 050. A tradução para essa linha é que a instrução BNE irá executar, a cada ciclo do clock, o que está na linha no bloco 1 (figura 03) primeiro e depois executará o que está no bloco 11 da mesma figura.

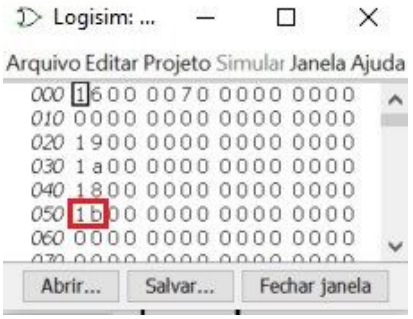
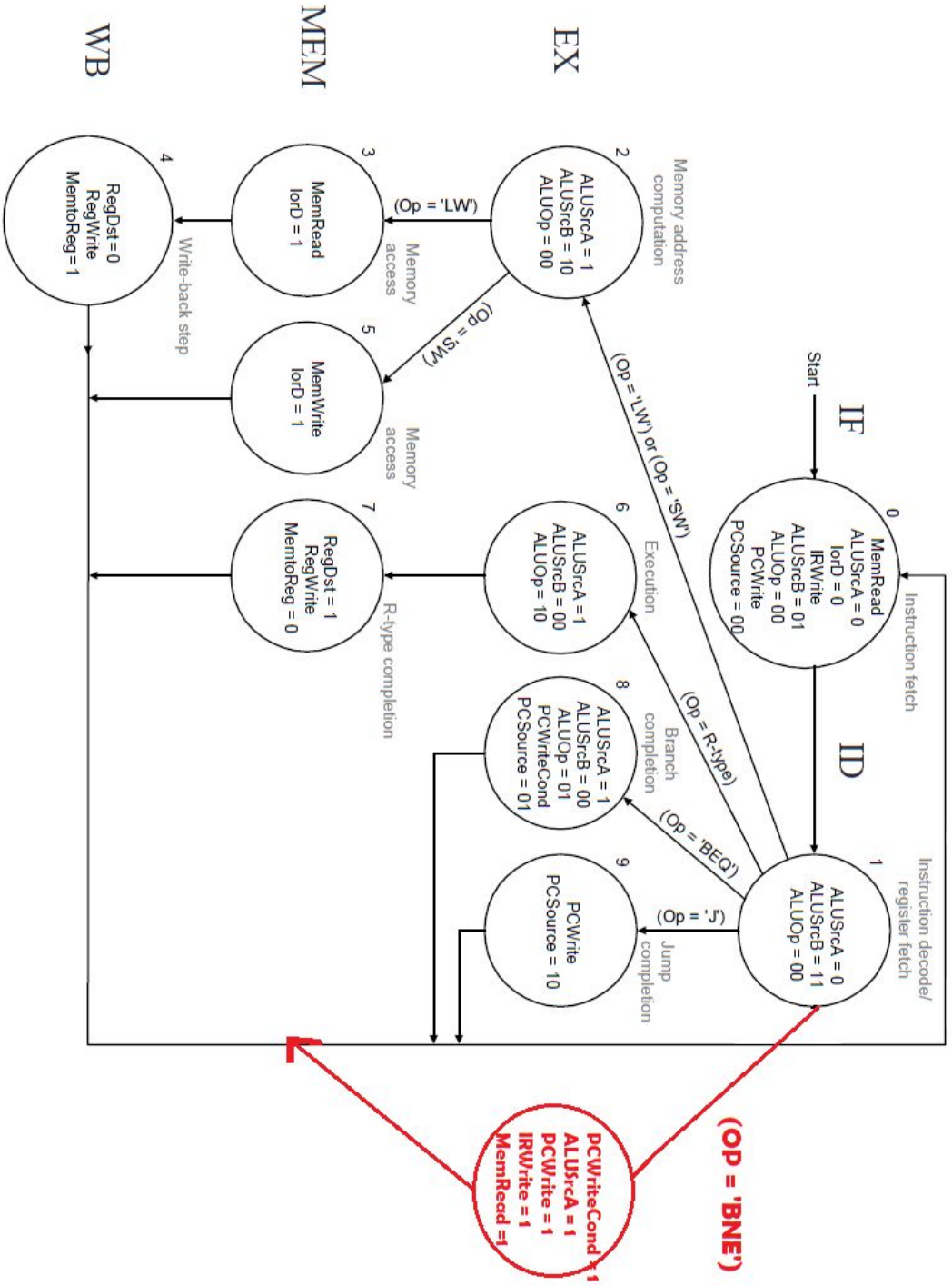


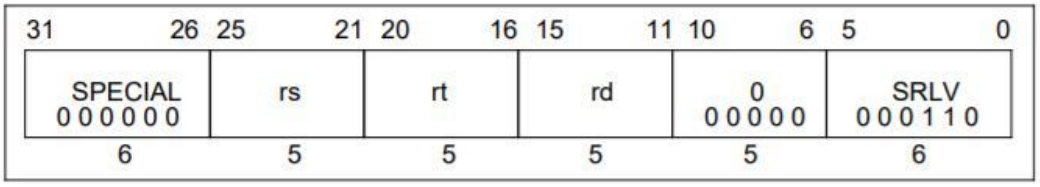
Figura 04: ordem em que cada instrução executa.

O diagrama de estados ficou da seguinte forma:



4. Implementação da instrução SRLV

SRLV Shift Right Logical Variable SRLV



Format:

SRLV rd, rt, rs

Description:

The contents of general register *rt* are shifted right by the number of bits specified by the low-order five bits of general register *rs*, inserting zeros into the high-order bits.

The result is placed in register *rd*.

In 64-bit mode, the operand must be a valid sign-extended, 32-bit value.

Operation:

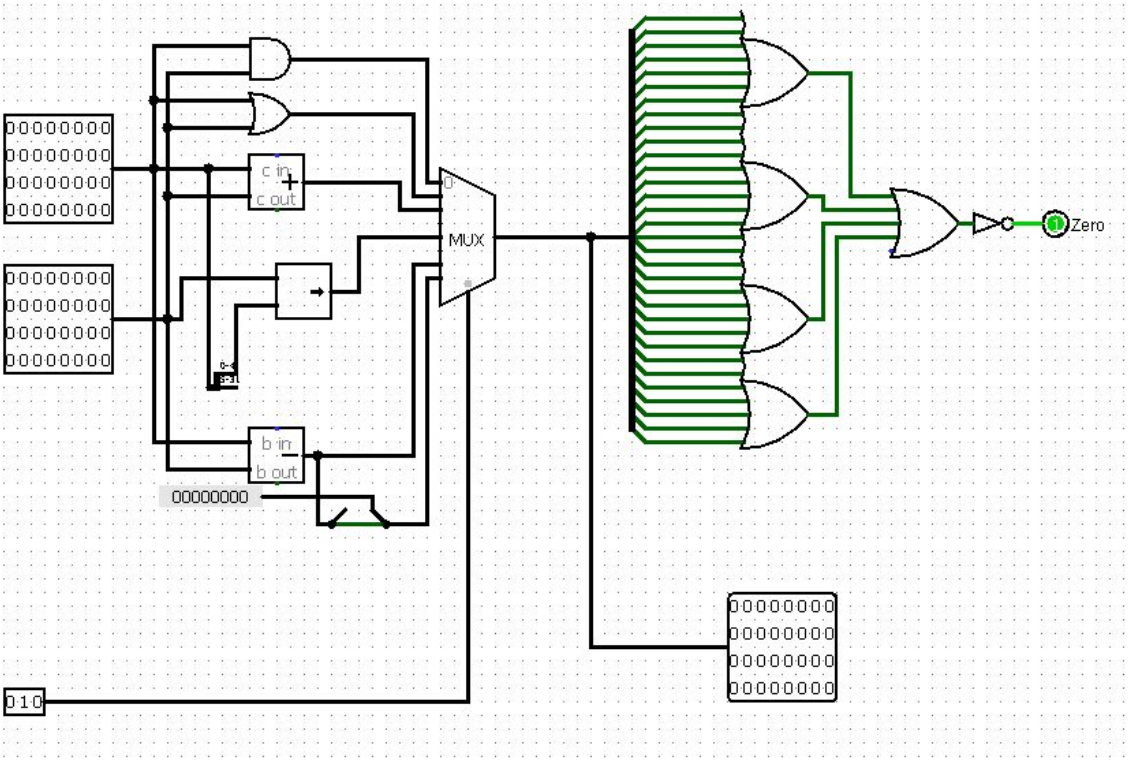
```
32   T:   s ← GPR[rs]4...0
          GPR[rd] ← 0^s || GPR[rt]31...s

64   T:   s ← GPR[rs]4...0
          temp ← 0^s || GPR[rt]31...s
          GPR[rd] ← (temp31)^32 || temp
```

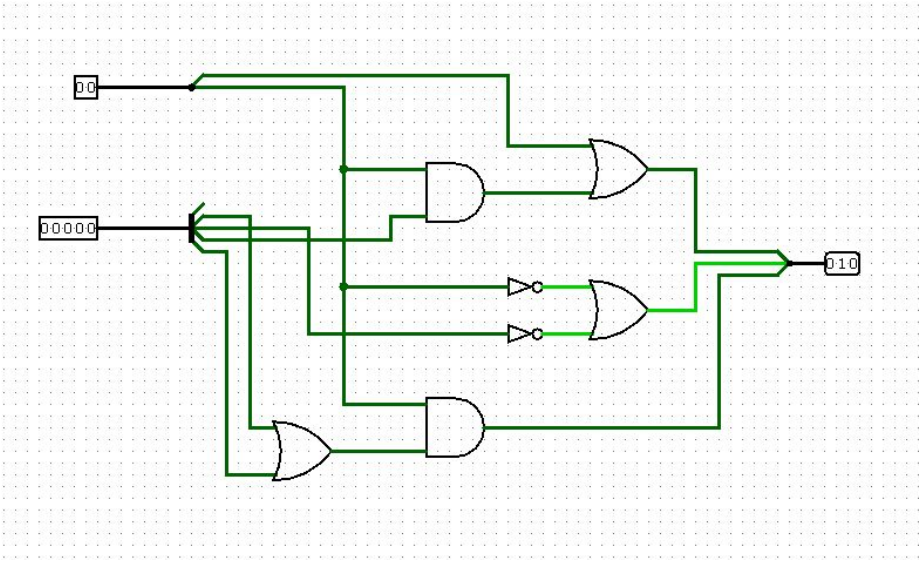
4.1 - Alterações no monociclo:

Para implementar a operação de shift right logical variable no processador MIPS monociclo foi acrescentada a operação de shift no interior na ULA. Todas as etapas (fetch, decode, operate e writeback) são idênticas à quaisquer outras instruções do tipo R, portanto não foram necessárias alterações nos sinais de controle.

Foi adicionado no interior da ULA um shifter que recebe como dado o conteúdo do registrador rt e como shift amount os cinco bits menos significativos do registrador rs. Após sair da ULA o resultado é gravado em rd. (Semelhante ao comportamento de todas as outras instruções do tipo R.)



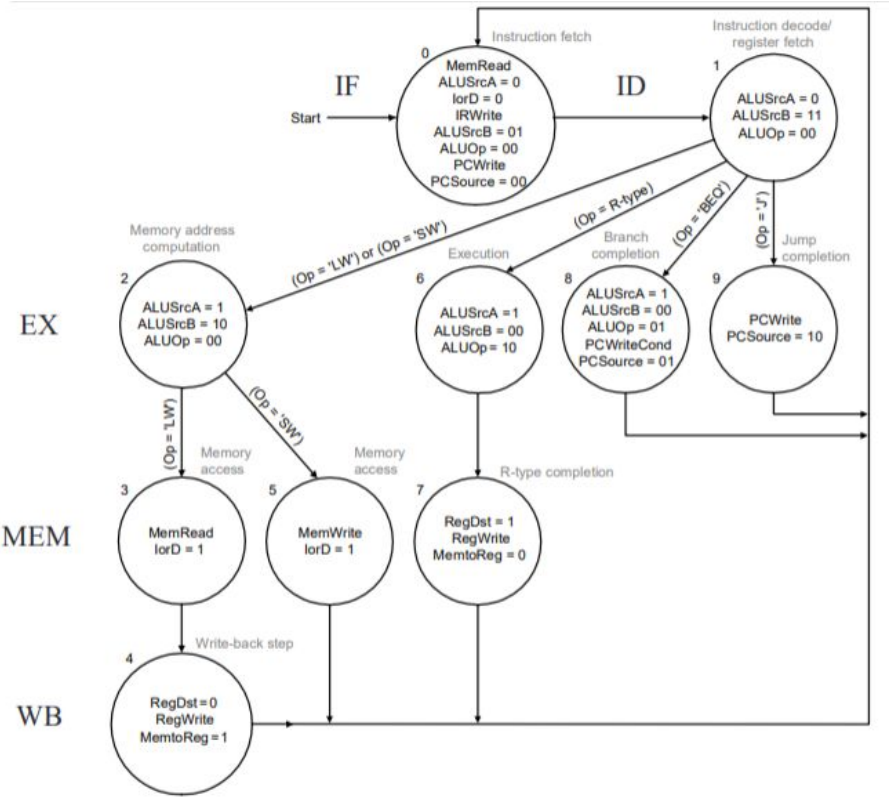
Tendo em vista que a operação realizada no interior da ULA é decidida pelos bits de saída da ALUControl que, por sua vez, recebe como entrada AluOp e o os cinco bits menos significativos do campo Funct da instrução; ao simular a situação em que temos como entrada AluOp = 10 (instruções tipo R) e o campo funct = 00110 (SRLV) temos como saída 100 (4). Por isso, o shifter está ligado à entrada correspondente ao número quatro no multiplexador da ULA.



4.2 - Alterações no multiciclo e Pipeline:

As alterações nas versões MIPS multiciclo e pipeline foram análogas às alterações realizadas no monociclo

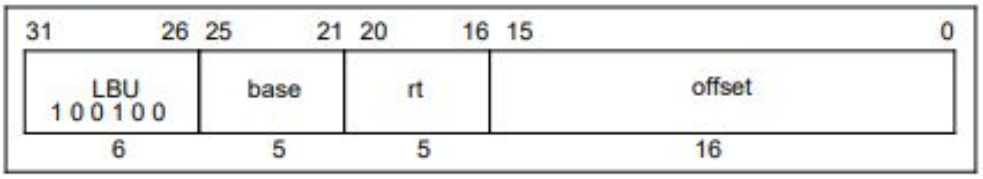
O diagrama de estados da FSM do MIPS multiciclo se mantém o mesmo e é o que segue:



.5. Implementação da instrução LBU

Appendix A

LBU — Load Byte Unsigned — LBU



Format:

LBU rt, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The contents of the byte at the memory location specified by the effective address are zero-extended and loaded into general register *rt*.

Operation:

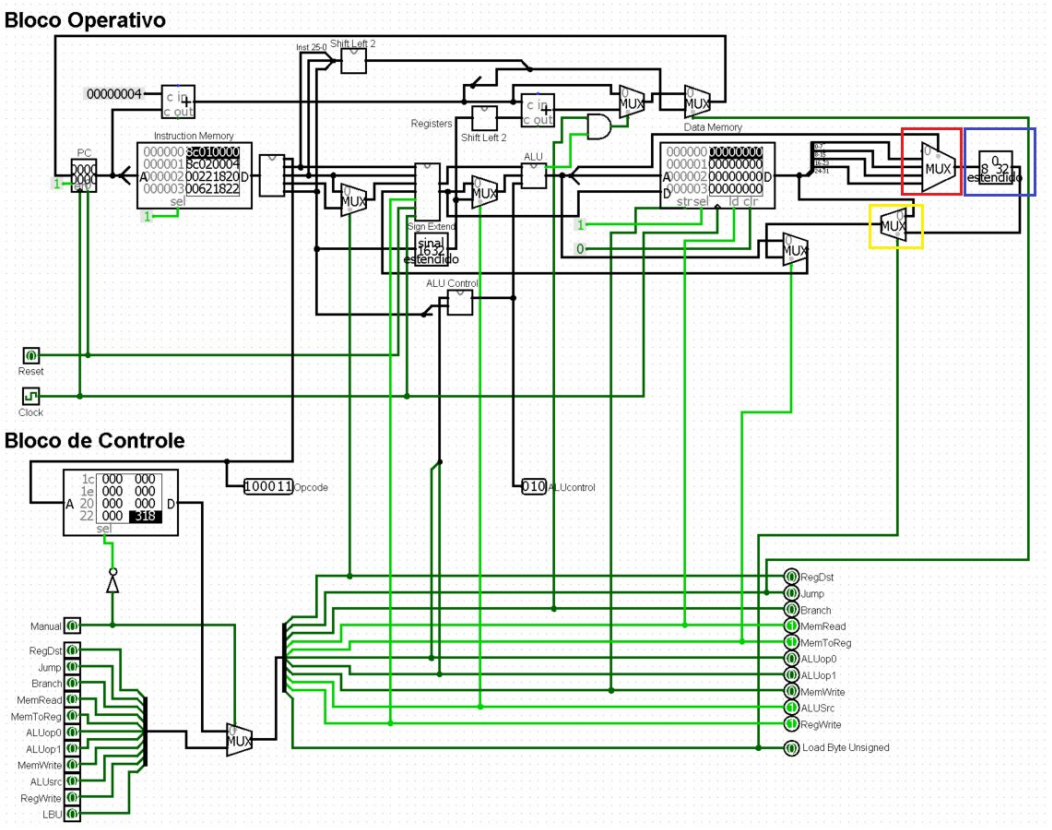
32 T:
vAddr ← $((offset_{15})^{16} \| offset_{15...0})$ + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
$pAddr \leftarrow pAddr_{PSIZE-1...3} \| (pAddr_{2...0}$ xor $ReverseEndian^3)$
mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
$byte \leftarrow vAddr_{2...0}$ xor $BigEndianCPU^3$
$GPR[rt] \leftarrow 0^{24} \| mem_{7+8*byte...8*byte}$

64 T:
vAddr ← $((offset_{15})^{48} \| offset_{15...0})$ + GPR[base]
(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
$pAddr \leftarrow pAddr_{PSIZE-1...3} \| (pAddr_{2...0}$ xor $ReverseEndian^3)$
mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
$byte \leftarrow vAddr_{2...0}$ xor $BigEndianCPU^3$
$GPR[rt] \leftarrow 0^{56} \| mem_{7+8*byte...8*byte}$

Exceptions:

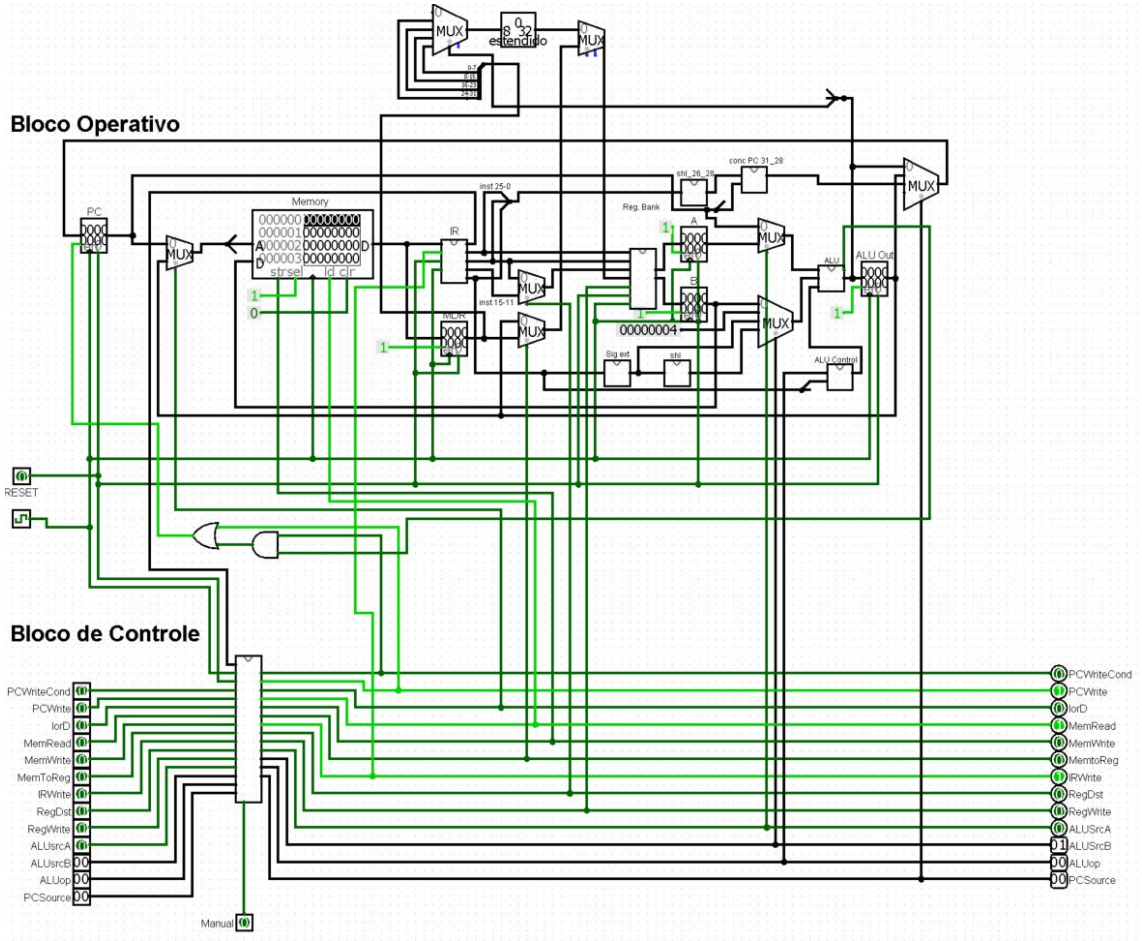
TLB refill exception — TLB invalid exception
Bus error exception — Address error exception

5.1 - Alterações no monociclo:

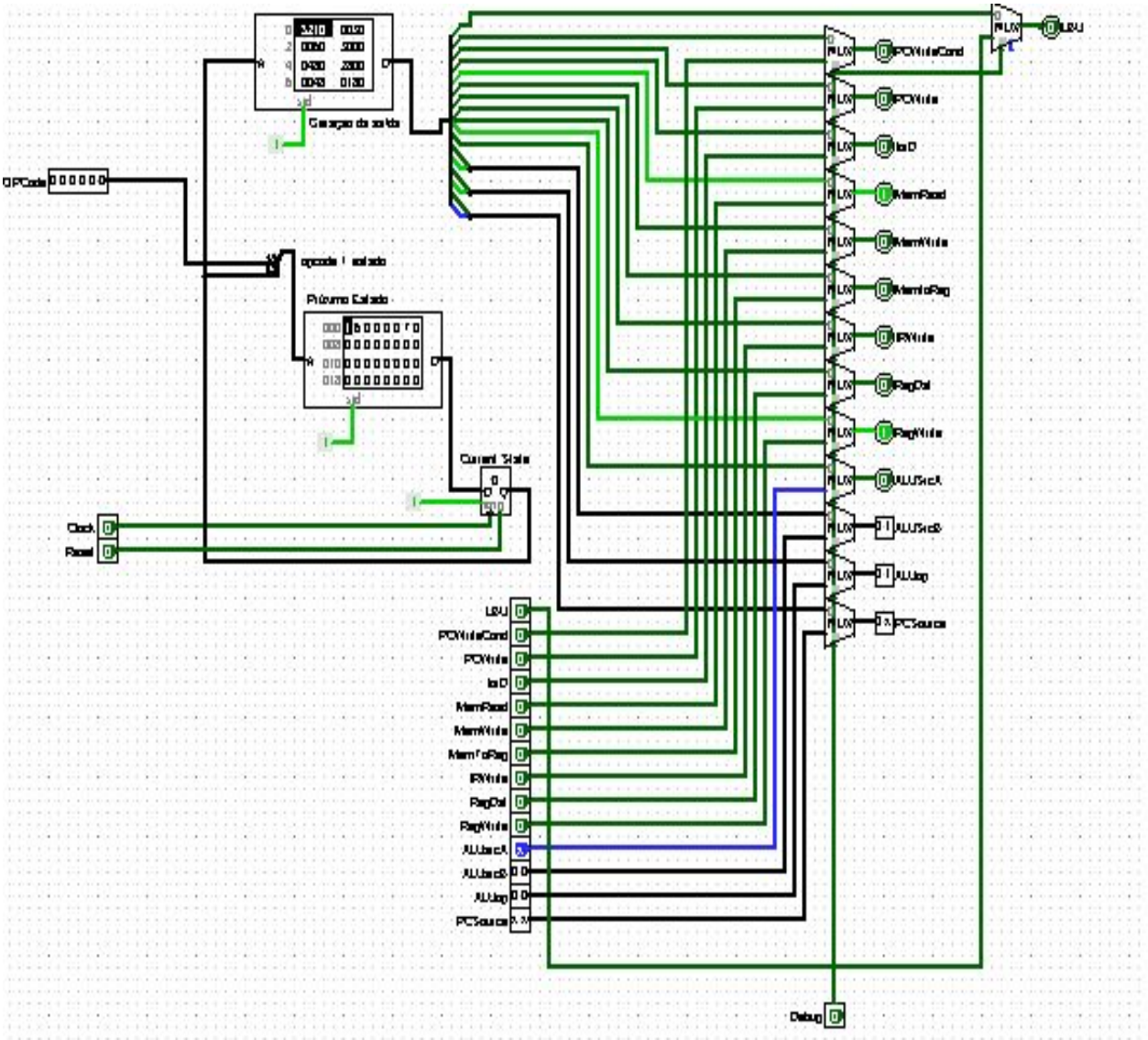


Foi adicionado um multiplexador(em vermelho) para decidir o bit menos significativo do load, uma caixa(em azul) para estender para 32 bits(zeros a esquerda) e um multiplexador(em amarelo) para decidir se a instrução é uma LBU com um sinal a mais no bloco de controle.

5.2 - Alterações no multiciclo:



No multiciclo foram feitas as mesmas alterações do monociclo, mudando apenas o lugar onde é feito o LBU. No monociclo foi feito na sequência do Data Memory e no multiciclo na sequência do RDM. No diagrama essa instrução seria feita no quarto ciclo MEM. O bloco de controle não ficou completo.


Geração da saída
OPCode
opcode + estado
Próximo Estado
Current State
Clock
Reset
LDU
PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemToReg
IRWrite
RegDst
RegWrite
ALUSrcA
ALUSrcB
ALUop
PCSource
Debug