UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL - UFRGS
INSTITUTO DE INFORMÁTICA
CIÊNCIA E ENGENHARIA DA COMPUTAÇÃO
ORGANIZAÇÃO DE COMPUTADORES - B
PROF. DR. ANTONIO CARLOS SCHNEIDER BECK FILHO

# RELATÓRIO DO TRABALHO 1:

## Implementação de novas instruções nos processadores MIPS.

EDUARDA PEREIRA, GABRIEL TAVARES, MAXIMUS DA ROSA E
NATHAN GUIMARÃES

PORTO ALEGRE/RS
2024

# 1. Instrução SRA (Shift Right Arithmetic)

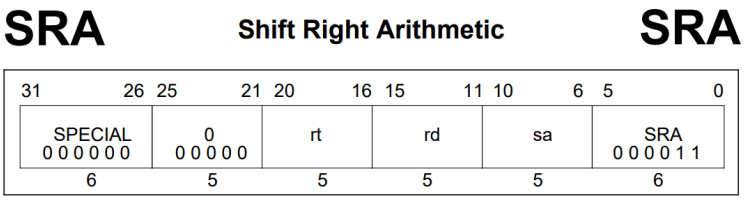



Figura 1: Formato da instrução SRA.

A instrução divide o *opcode* "000000" com as demais que fazem uso da Unidade Lógica Aritmética (ULA), sendo diferenciada apenas pelos 6 últimos *bits*, apelidados de campo *funct*. Por isso, a implementação do SRA visa a não alterar os sinais/*flags* de controle já feitos para as instruções de ADD e SUB, já que o caminho que os dados irão percorrer é o mesmo.

## A) Implementação no Monociclo

Foi necessário fazer modificações nas seguintes áreas do processador:

- Decodificador de instruções

  Pela Figura 1, nota-se que a instrução prevê a utilização de um campo que não era registrado na versão original do processador (os *bits* 10-6, chamados de *shift amount* (*sa*)). Para isso, uma nova saída foi adicionada no decodificador, para permitir que tenhamos acesso a esses *bits* e manipulá-los na ULA posteriormente.

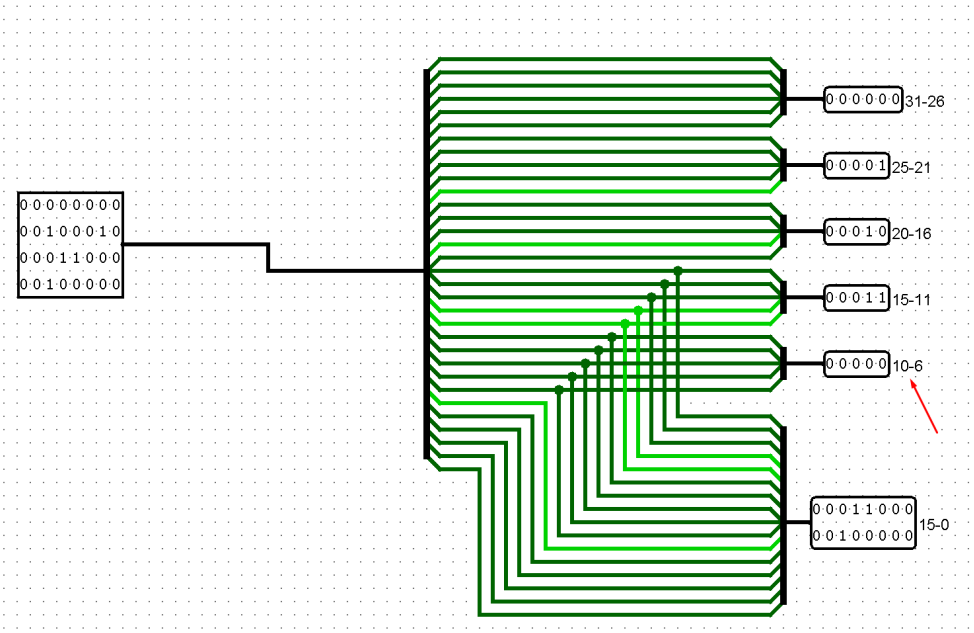



Figura 2: O decodificador de instruções modificado. Destacada pela seta vermelha está a nova saída adicionada.

- Unidade lógica e aritmética (ULA)

A implementação fornecida no *moodle* da disciplina deixava livre as entradas (respectivas aos *bits* de controle) “011”, “100” e “101” do multiplexador (MUX) da ULA livres. Por praticidade, o *hardware* necessário para o deslocamento foi adicionado na entrada “011” do MUX, por ser a primeira livre.

Dessa forma, os *bits* do *shift amount* saem do decodificador de instruções e vão diretamente para a ULA, por uma nova entrada que foi adicionada. Pela forma como já está implementado o *datapath*, os dados do registrador que deve ser deslocado já estarão presentes na segunda entrada de 32 *bits* da ULA, sem que haja necessidade de se modificar nada.

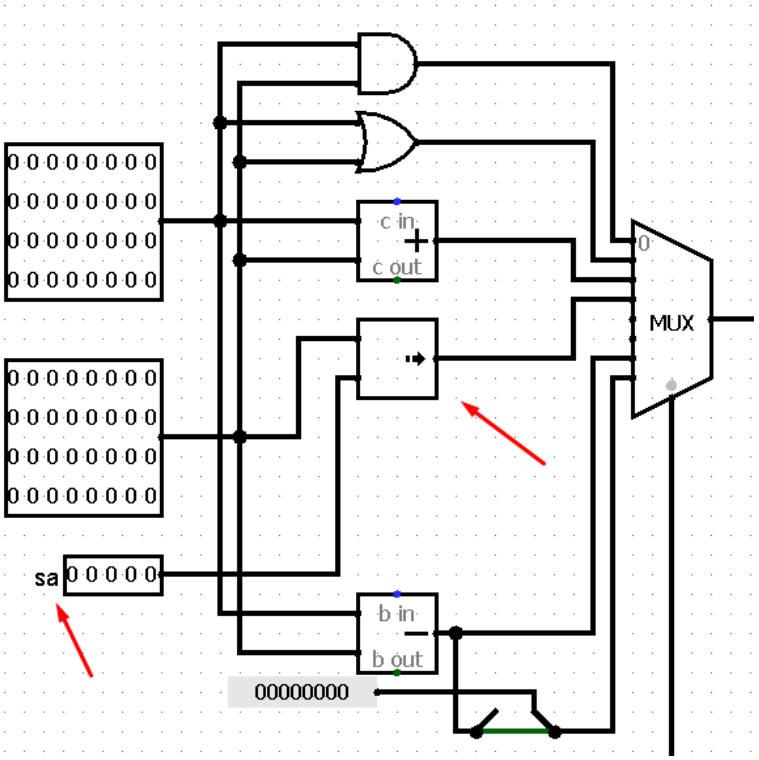



Figura 3: A ULA modificada. Destacados pelas setas vermelhas estão a nova entrada “sa” e o deslocador aritmético para a direita.

- Controle da ULA

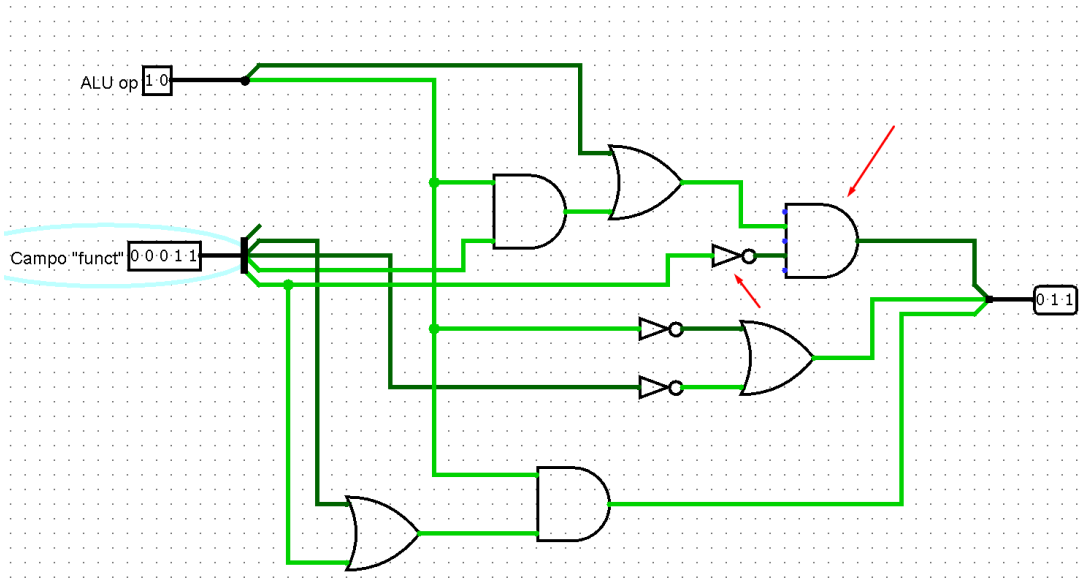
Como foi escolhido que a instrução SRA irá ser executada conforme os *bits* de controle “011” da ULA, precisa-se garantir que, ao receber o *opcode* “000000” (referente a instruções da ULA) e campo *funct* “0011” (os 4 *bits* menos significativos da instrução), a saída da ULA será o SRA. Na Tabela 1, é possível ver como deve ser implementado este novo controle. Ou seja, a intenção é fazer com que as instruções previamente adicionadas continuem funcionando, apenas adicionando uma nova possível saída.

| **Instrução** | **ALUOp** | | **Campo "funct"** | | | | **Saída** | | | **Operação na ULA** |
|---|---|---|---|---|---|---|---|---|---|---|
| LW / SW | 0 | 0 | X | X | X | X | 0 | 1 | 0 | ADD |
| BEQ | X | 1 | X | X | X | X | 1 | 1 | 0 | SUB |
| ADD | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 0 | ADD |
| SUB | 1 | X | 0 | 0 | 1 | 0 | 1 | 1 | 0 | SUB |
| SRA | 1 | X | 0 | 0 | 1 | 1 | 0 | 1 | 1 | SRA |
| AND | 1 | X | 0 | 1 | 0 | 0 | 0 | 0 | 0 | AND |
| OR | 1 | X | 0 | 1 | 0 | 1 | 0 | 0 | 1 | OR |
| SOLT | 1 | X | 1 | 0 | 1 | 0 | 1 | 1 | 1 | SOLT |

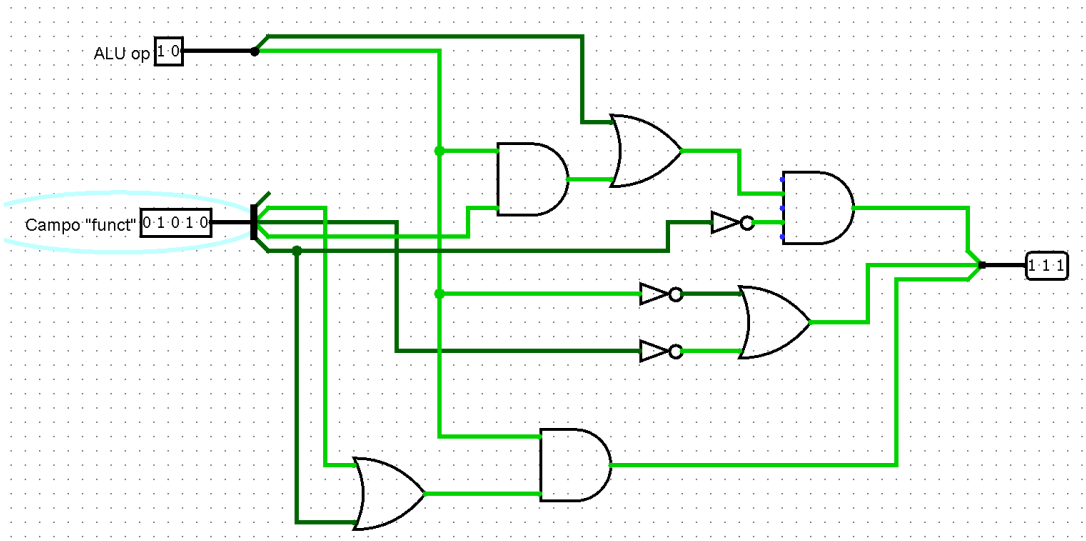
Tabela 1: A nova tabela-verdade do controle da ULA. Em destaque, a linha adicionada para suprir a nova instrução.

Sem alterar nada na forma como foi implementado o controle, as entradas respectivas às instruções SRA e Set-On-Less-Than (SOLT) teriam a mesma saída de *bits* “111”. Como as entradas do campo *funct* de ambas difere tanto no *bit* mais significativo quanto no *bit* menos significativo (“0011” e “1010”, respectivamente em destaque) , qualquer um dos dois poderia ser usado para modificar o controle. O *bit* menos significativo foi escolhido como critério de diferenciação entre as duas instruções, utilizando-o para garantir que a saída do SRA seja “011”.

Como o *bit* mais significativo da saída ficava constante em “1” para ambas as entradas, ao adicionar uma porta *AND* deste sinal com o inverso do *bit* menos significativo do campo *funct*, cria-se uma segunda tabela-verdade que nos permite que a saída do SOLT se preserve como “111”, mas que a saída do SRA fique “011”, elucidado na Figura 4.



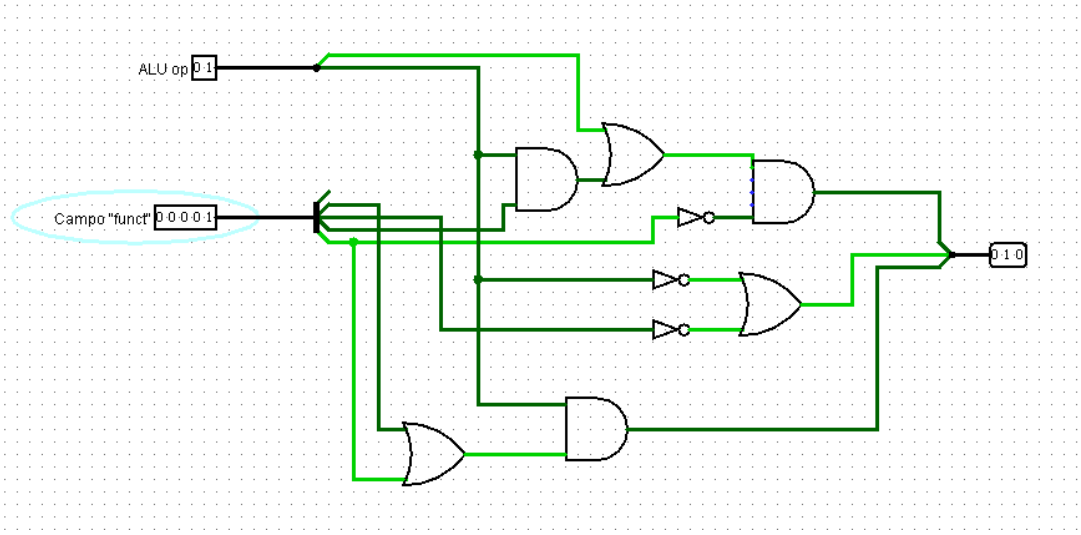

(a) Entradas e saídas do controle para a instrução SRA.



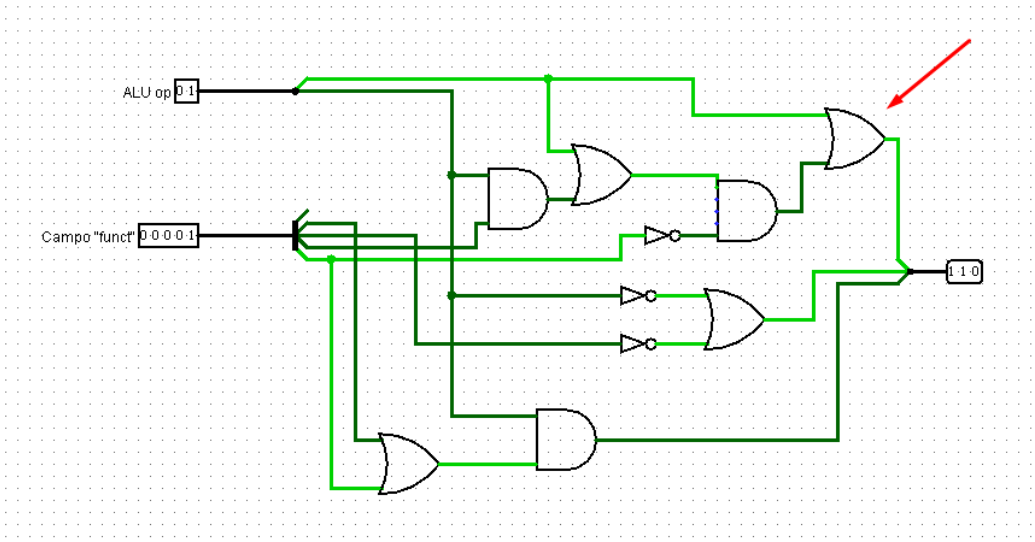

(b) Entradas e saídas do controle para a instrução SOLT.

Figura 4: O controle da ULA modificado. As setas vermelhas indicam as portas que foram adicionadas.

Contudo, detectamos posteriormente que as alterações feitas causavam prejuízo à instrução BEQ a depender do "lixo" de memória que estivesse presente no campo *funct*. A saída esperada para ALUOP = "01" é sempre constante e igual a "110" (instrução de SUB) independentemente do campo *funct*, mas o mesmo não ocorria quando este tinha como entrada "0001", mostrado na Figura 5. Portanto, foi adicionada uma porta lógica do tipo *OR*, destacada na mesma Figura, para corrigir esta falha. Dessa forma, todas as instruções da Tabela 1 têm suas saídas devidamente respeitadas.




(a) Falha constatada para a instrução BEQ.




(b) Saída corrigida para a instrução BEQ.

Figura 5: O controle da ULA final. A seta vermelha indica a porta que foi adicionada por último.

## B) Implementação no Multiciclo

As mudanças foram feitas nos mesmos lugares e da mesma forma que a implementação no Monociclo, anteriormente explicada. Ou seja, foram adicionadas uma saída para o campo de *shift amount* no decodificador de instruções, um bloco de *hardware* que implementa propriamente o *shift* aritmético para a direita dentro da ULA (conectado novamente na entrada "011" do MUX) e a mesma alteração foi feita no controle da ULA (já que a mesma função vista na Tabela 1 está sendo implementada).

Como somente o *hardware* do decodificador de instruções é diferente entre as áreas citadas dos dois processadores, a sua modificação está mostrada na Figura 5. Ou seja, as

modificações feitas na ULA e em seu respectivo controle para a versão multiciclo do MIPS são exatamente iguais às da versão monociclo e podem ser vistas nas Figuras 3 e 4, respectivamente.

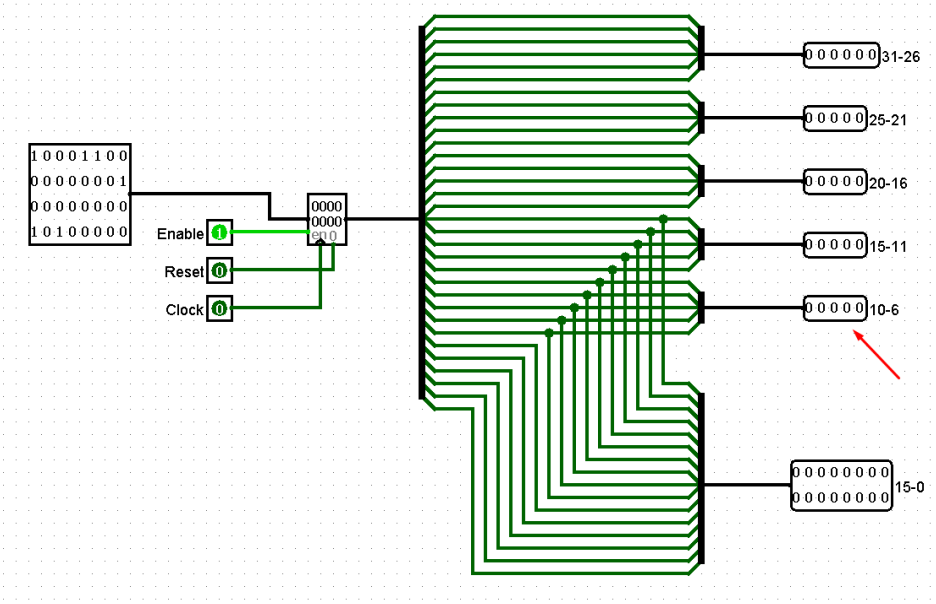



Figura 6: O decodificador de instruções modificado. Destacada pela seta vermelha está a nova saída adicionada.

## C) Implementação no Pipeline

Novamente, as mesmas alterações apresentadas nas Figuras 3, 4 e 5 foram feitas (o decodificador de instruções do pipeline se assemelha ao do multiciclo, mas a organização interna da ULA e de seu controle são iguais nos três processadores). Há apenas uma novidade, relacionada ao *datapath* do processador: como existe um estágio de pipeline entre o decodificador de instruções e as execuções na ULA, foi necessário adicionar um novo registrador que guarda o valor do *shift amount* (*bits* 10-6) para que ele seja utilizado no próximo estágio dentro da ULA. A Figura 6 ilustra onde este novo registrador se encontra, logo abaixo dos demais que guardam as saídas do decodificador de instruções.

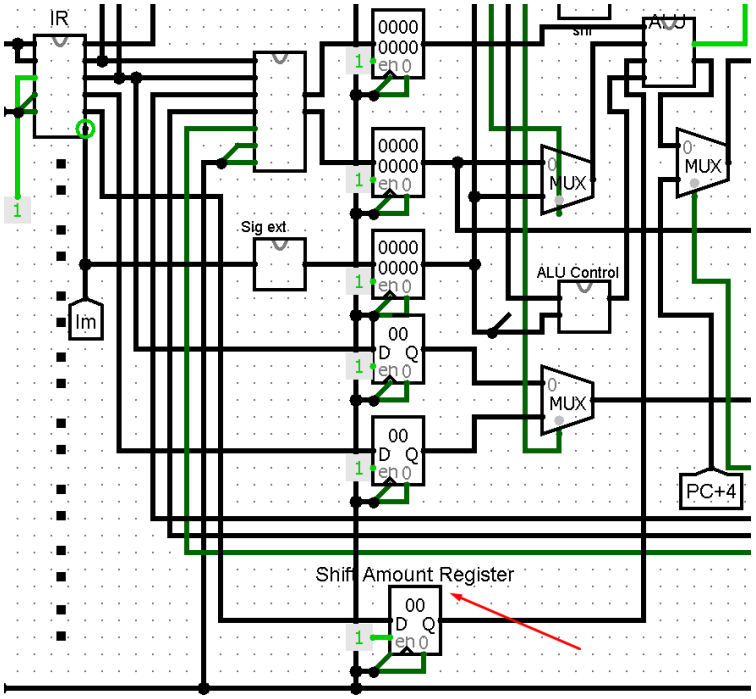



Figura 7: O *datapath* modificado. Destacado pela seta vermelha está o novo registrador adicionado.

## 2. Instrução JAL (Jump And Link)

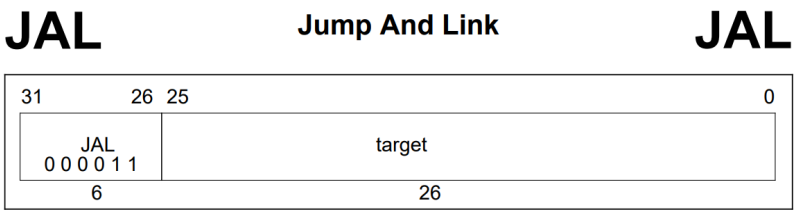



Figura 8: Formato da instrução JAL.

### A) Implementação no Monociclo

Esta instrução aproveita partes do hardware da instrução JUMP , incorporando novos componentes para armazenar o valor do Contador de Programa (PC) adicionado de 4 unidades no registrador 31.

- Registrador 31 como opção de Write Address

  Para o valor de PC + 4 ser registrado no registrador 31, é necessário que este registrador seja uma opção de registro, além dos registradores rd e rt. Para isso, foi adicionado um multiplexador, que deve escolher entre um dos registradores comuns (rd e rt) e o registrador 31.

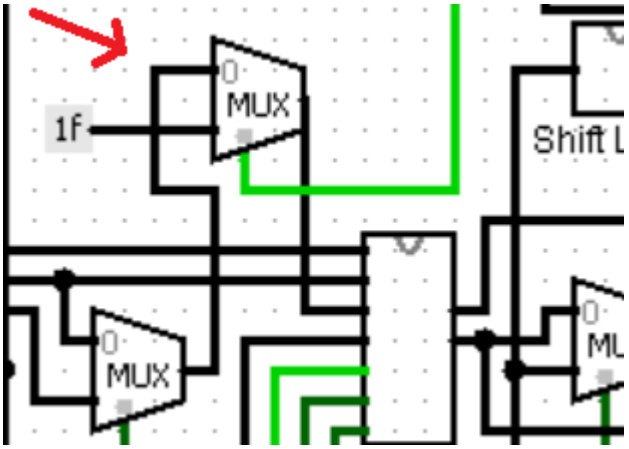



Figura 9: Multiplexador adicionado ao circuito destacado pela seta vermelha.

- Enviar PC + 4  ao banco de registradores como Write Data

Por fim, um multiplexador foi adicionado após a memória de dados, para que se possa repassar o valor do PC + 4 para o banco de registradores.

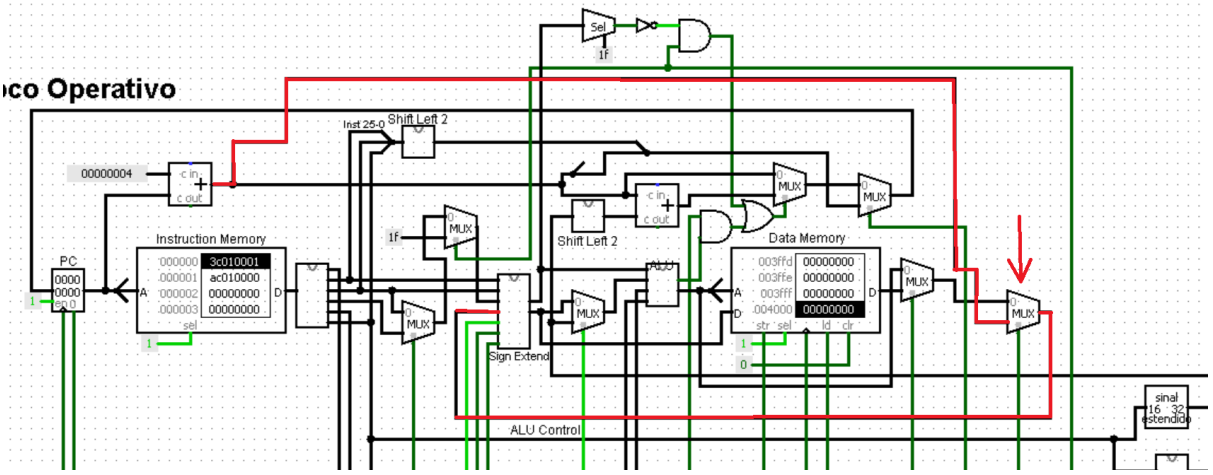



Figura 10: Caminho que leva “PC + 4” a ser registrado no banco de registradores, destacado em vermelho. A seta vermelha destaca o novo multiplexador adicionado.

- Novo sinal de controle JalEnable

Para esta instrução, foi criado um novo sinal de controle chamado de “JalEnable”. Este sinal de controle é enviado para os dois multiplexadores adicionados ao circuito (explicados acima). A função deste novo sinal, é habilitar a passagem do registrador 31 como o registrador a ser escrito e habilitar o valor de PC + 4 como dado a ser escrito no registrador 31. É evidente que para essa adição, é necessário fazer alterações nos distribuidores.

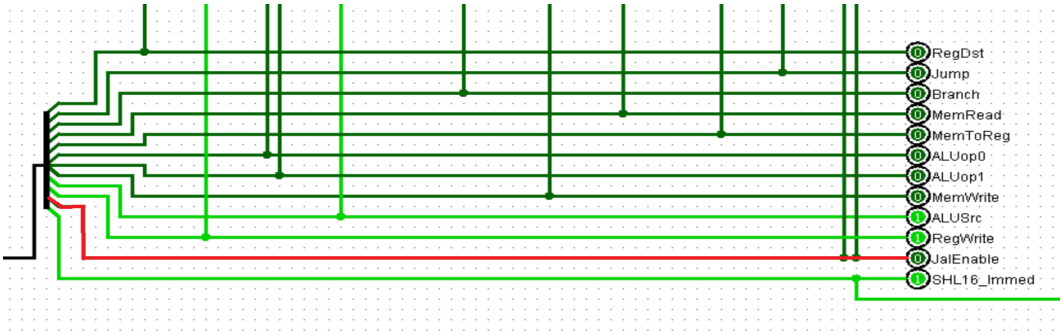


Figura 11: Novo sinal de controle JalEnable destacado em vermelho.

- Modificação na ROM

Como um novo sinal de controle foi adicionado, foi necessário fazer uma modificação na ROM. Como o opcode da instrução JAL é igual a 000011 (3 em binário), foi necessário fazer uma modificação na terceira posição da tabela (contando a partir de 0). Esta modificação foi feita a partir dos sinais de controle, pensando em quais deveriam estar ativados e quais não deveriam estar ativados. A figura abaixo mostra a ordem em que os sinais de controle estão organizados, sendo SHL16_Immed o sinal de controle referente ao bit mais significativo e RegDst o sinal de controle referente ao bit menos significativo:

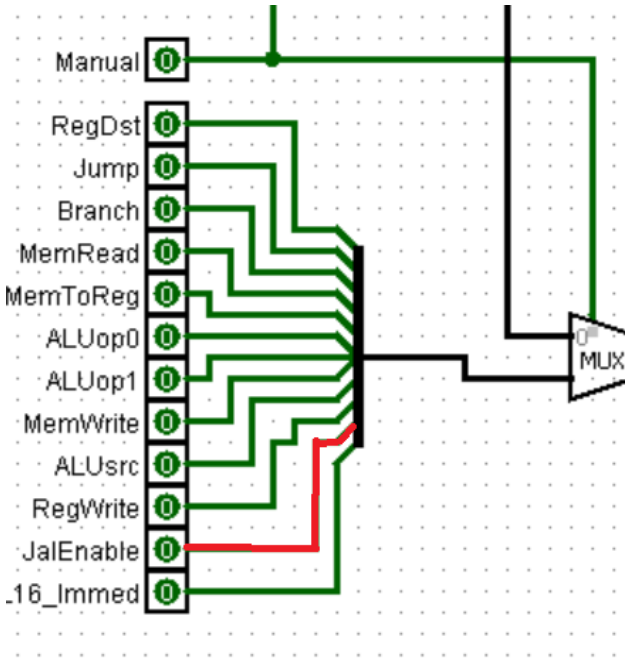



Figura 12: sinais de controle do processador MIPS monociclo.

Assim, como os sinais de controle JalEnable, RegWrite e Jump devem estar em 1, o número em binário que codifica esses sinais de controle será 011000000010, que é igual a 602 em hexadecimal. Portanto, adicionamos o número 602 na terceira posição da tabela da ROM:

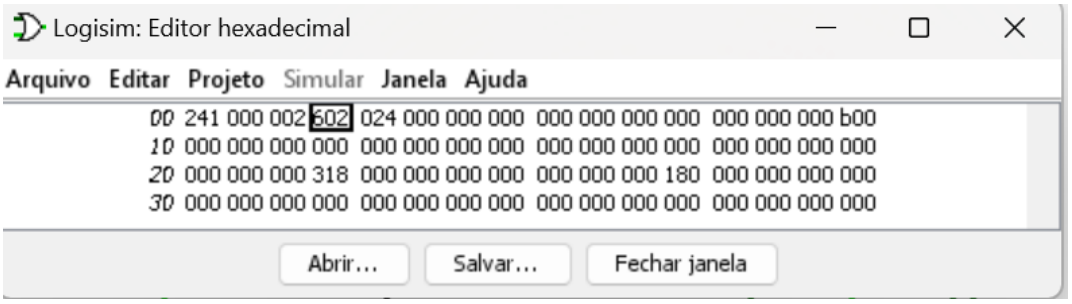



Figura 13: ROM do processador MIPS monociclo modificada.

## B) Implementação no Multiciclo

Para o Multiciclo, as adições no hardware foram idênticas às adições no monociclo, sendo adicionados dois multiplexadores: um para escolher o registrador 31 como Write Address e o outro para escolher PC + 4 como Write Data:

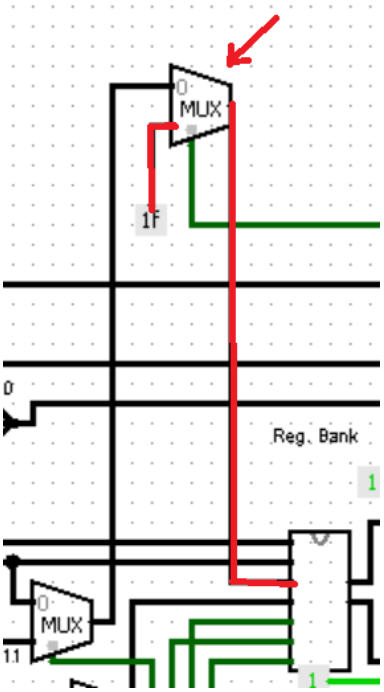



Figura 14: caminho que escolhe o registrador 31 como Write Address no banco de registradores. Multiplexador adicionado ao circuito destacado por uma seta vermelha.

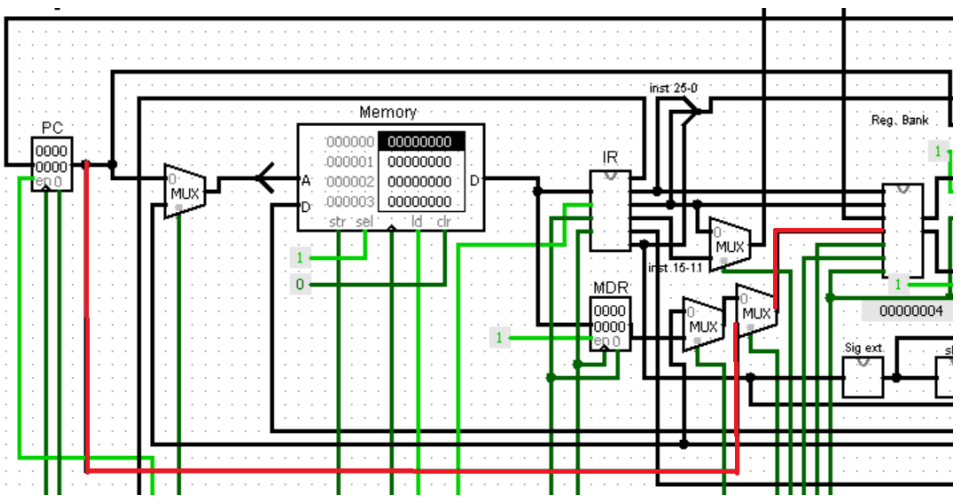


Figura 15: caminho que seleciona PC + 4 como Write Data.

- Novo sinal de controle JalEnable

Para o multiciclo, também foi criado um novo sinal de controle JalEnable, que é enviado para os 2 multiplexadores adicionados ao circuito. É evidente que para essa adição, é necessário fazer alterações nos distribuidores do circuito.

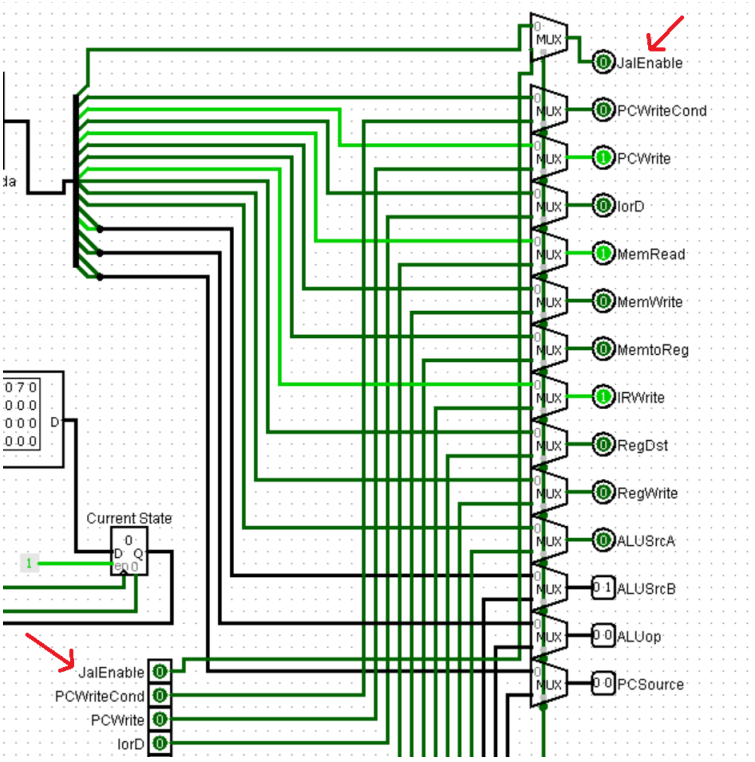



Figura 16: Imagem de dentro do bloco FSM_ROM. Novo sinal JalEnable destacado por setas vermelhas.

- Modificação na ROM

Na tabela da ROM para o processador multiciclo, cada coluna representa um estado diferente. Dessa forma, foi criado um novo estado (estado 10), para suportar a nova instrução. No estado 10, os seguintes sinais são diferentes de 0: PcSource (deve ser igual a 10), RegWrite (deve ser igual a 1), PcWrite (deve ser igual a 1) e JalEnable (deve ser igual a 1). Codificando para binário e olhando a ordem dos sinais de controle na figura X, temos o número 1 0100 0000 1000 0010, que é igual a 14082 em hexadecimal. Assim, na posição 10 da tabela (contando a partir de 0), registramos o número 14082.


Figura 17: tabela do bloco ROM.

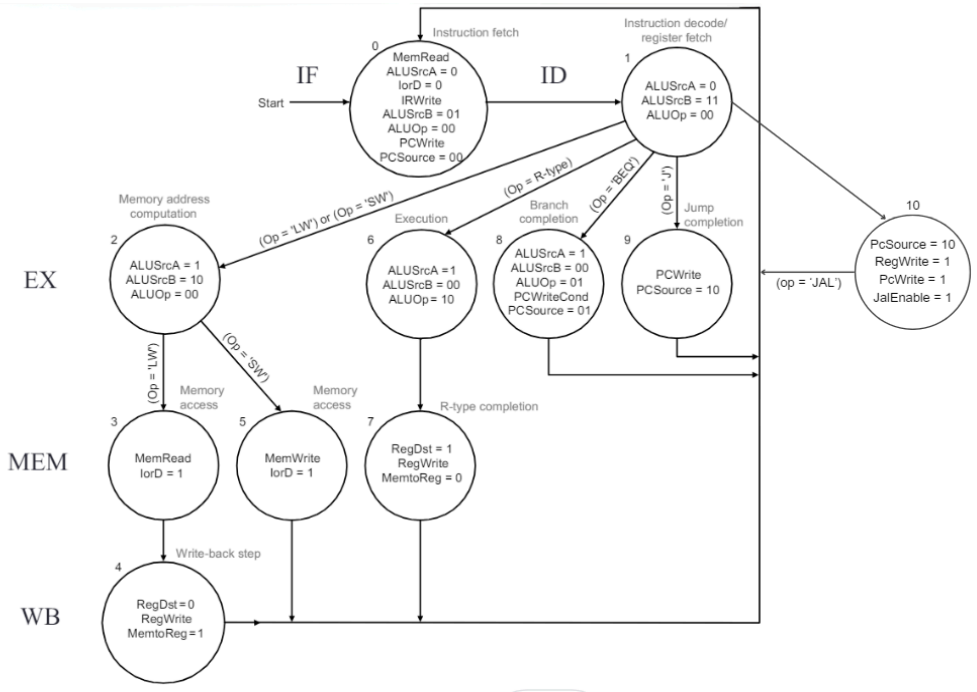



Figura 18: Diagrama de estados com estado “10” adicionado.

- Mudança na lógica de próximo estado

Na tabela da lógica de próximo estado, cada linha representa instruções e cada coluna representa qual é o próximo estado da determinada instrução. Como a instrução JAL tem opcode 000011, a linha na tabela que representa esta instrução é a linha de número 030. Assim, nesta linha colocamos os estados 1 e 10 (“a” em hexadecimal), indicando os estados pelo qual a instrução deve passar durante a sua execução (lembrando que todas as instruções passam pelo estado 0 anteriormente).

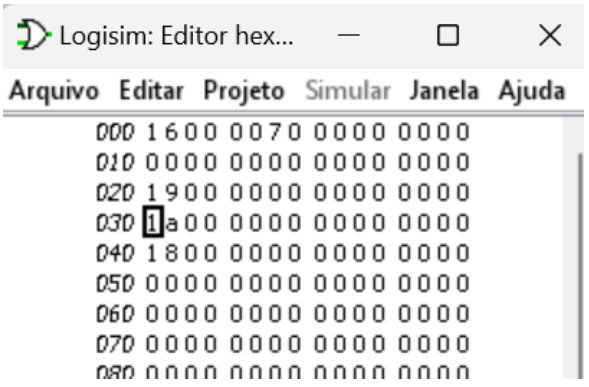



Figura 19: tabela da lógica de próximo estado modificada na linha 030.

## C) Implementação no Pipeline

Para o Pipeline, as adições no hardware foram idênticas às adições no monociclo e no multiciclo, sendo adicionados dois multiplexadores: um para escolher o registrador 31 como Write Address e o outro para escolher PC + 4 como Write Data.

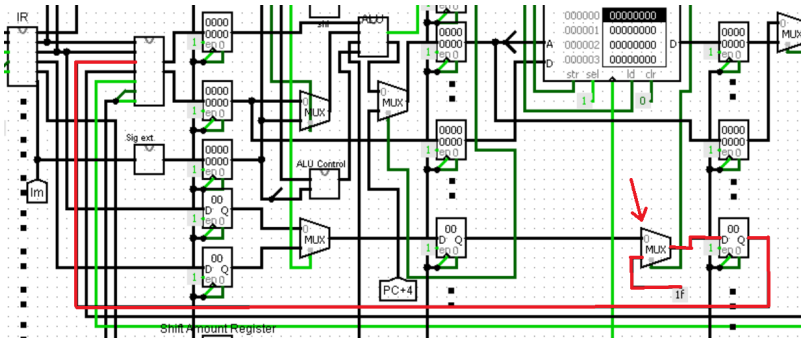



Figura 20: caminho que leva registrador 31 como Write Adress ao banco de registradores.

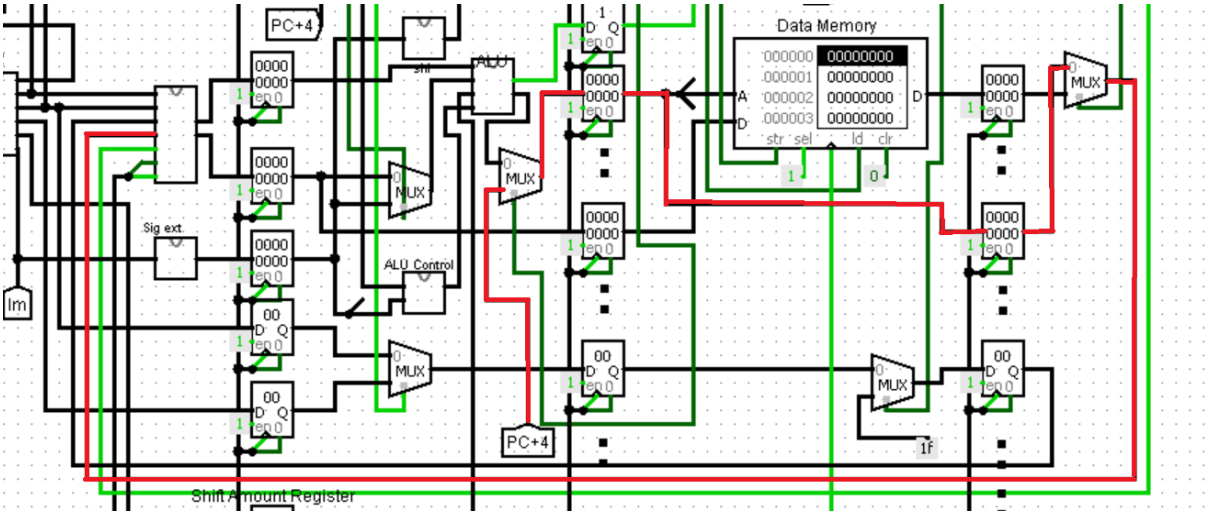



Figura 21: caminho que leva “PC + 4” como Write Data ao banco de registradores.

- Novo sinal de controle JalEnable

Assim como nos outros processadores, também foi necessário adicionar um novo sinal de controle JalEnable.

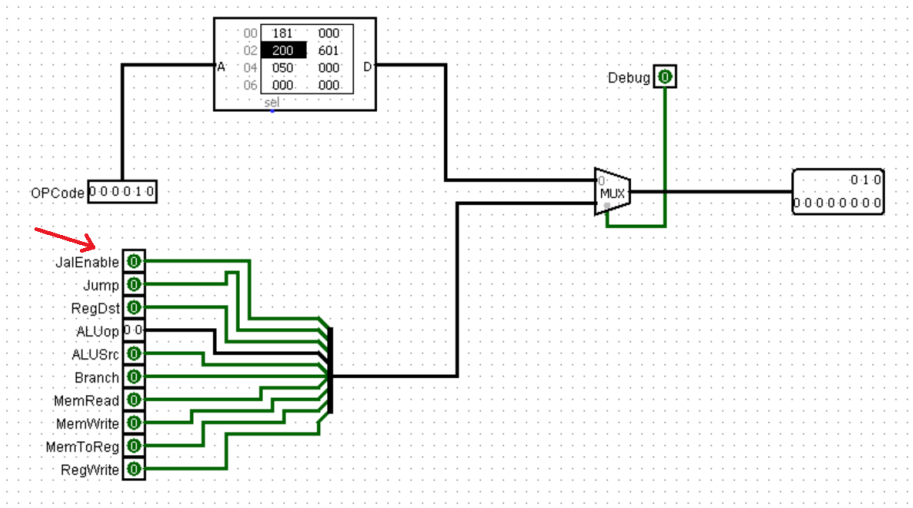


Figura 22: imagem de dentro do bloco da ROM com novo sinal JalEnable.

Esse novo sinal foi distribuído nas etapas 3 e 4 do Pipeline, sendo necessário realizar mudanças nos distribuidores.

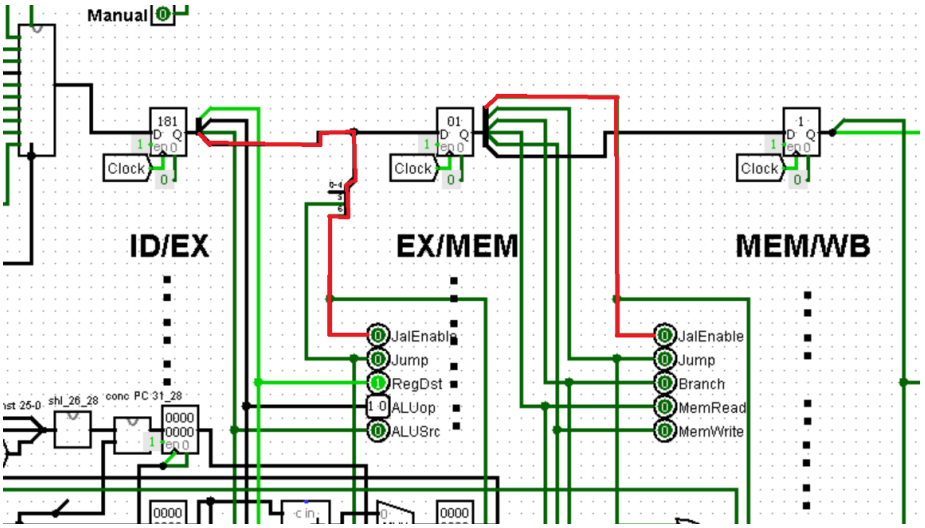



Figura 23: imagem do sinal de controle saindo dos registradores e sendo distribuído para as etapas 3 e 4.

- Mudança na ROM

Na terceira posição da tabela (contando a partir de 0) da ROM foi adicionado o número 601 em hexadecimal (que representa o número 0110 0000 0001 em binário), para ligar os sinais de controle JalEnable, Jump e RegWrite em 1.

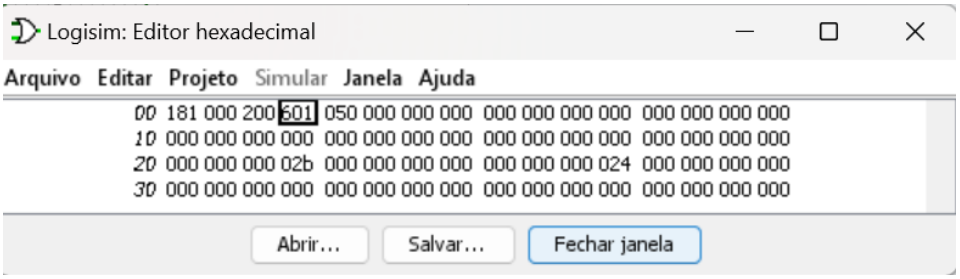



Figura 24: tabela da ROM atualizada com o valor 601 na terceira posição.

# 3. Instrução LUI (Load Upper Immediate)

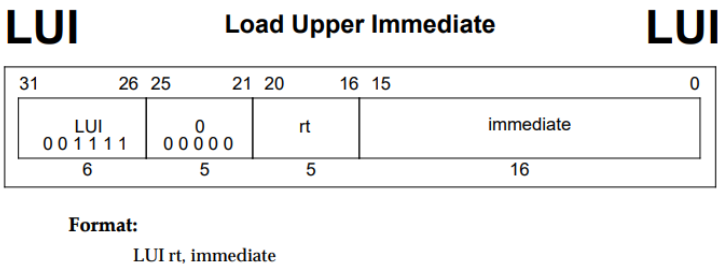



Figura 25: Formato da instrução LUI.

A instrução LUI carrega o valor imediato nos 16 bits mais significativos do registrador rt, e zera os 16 bits menos significativos. Dentro do circuito, o caminho percorrido é semelhante ao de um "LW rt, x($0)", com x = shl16(immediate). No entanto, ao invés de utilizar o valor de (offset + base) como um endereço na memória, esse valor é salvo diretamente no registrador rt.

## A) Implementação no Monociclo

Esta instrução aproveita partes do hardware da instrução LW, incorporando novos componentes para que o valor repassado à segunda entrada da ULA seja o do campo 'immediate' deslocado 16 vezes para a esquerda, compondo um novo número de 32 bits.

- SHL 16 ('immediate')
  Foi adicionado um novo deslocador lógico para a esquerda, com 32 bits de dados, para deslocar o valor com sinal estendido 16 casas para a esquerda, para obtermos o efeito "Upper Immediate".

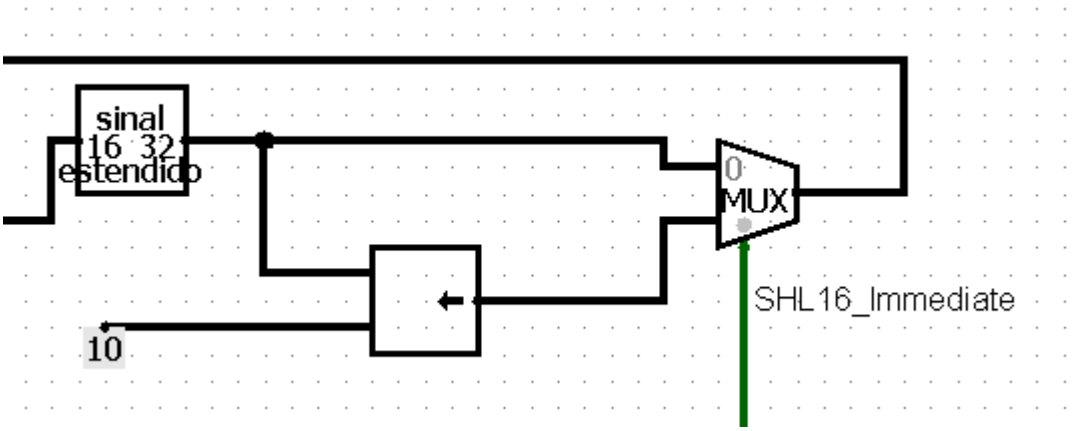



Figura 26: Deslocador adicionado ao circuito.

- SHL 16 (‘immediate’) como opção de entrada para a ULA
  Foi adicionado um novo mux, para selecionar o que viria na segunda entrada do mux controlado pelo sinal “ALUSrc”, que é responsável por decidir se o valor da segunda entrada da ULA virá de forma imediata ou por meio de um registrador.

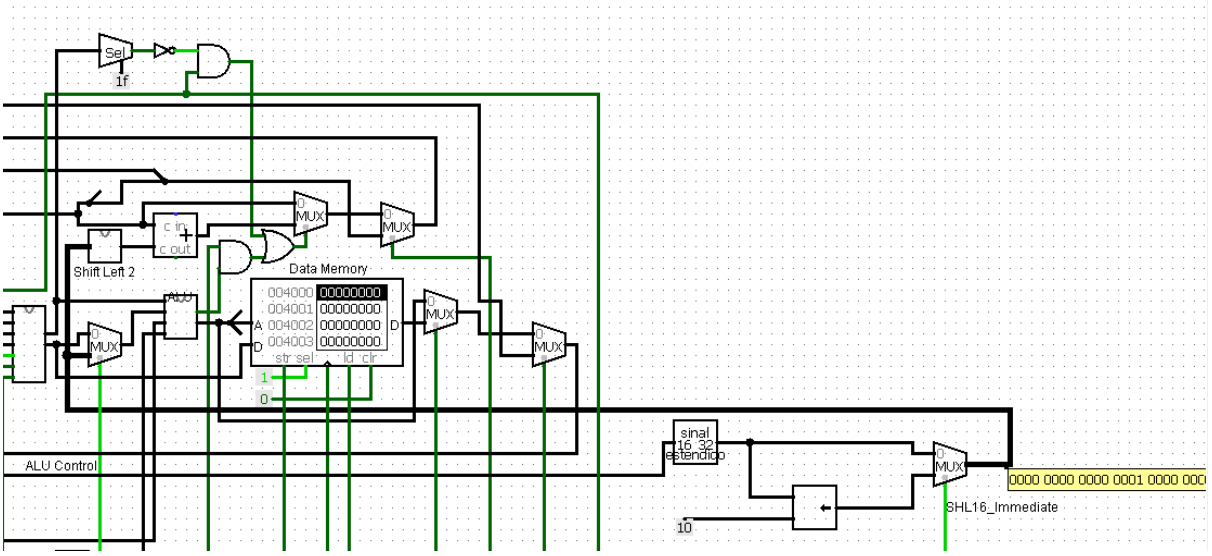



Figura 27: Caminho percorrido pela saída do mux.

- Novo sinal de controle SHL16_Immed

  Para esta instrução, foi criado um novo sinal de controle chamado de “SHL16_Immed”. Este sinal de controle é enviado para o multiplexador adicionado ao circuito (explicado acima). A função deste novo sinal, é escolher entre a concatenação de 16 bits de sinal estendido + bits[15-0] da instrução ou bits[15-0] da instrução + 16 * 0.

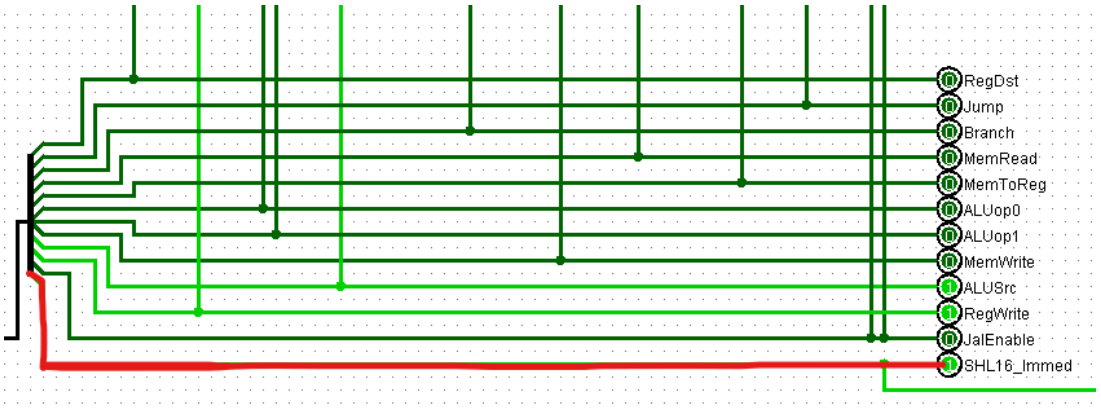



Figura 28: Novo sinal de controle SHL16_Immed destacado em vermelho.

- Modificação na ROM

Como um novo sinal de controle foi adicionado, foi necessário fazer uma modificação na ROM. Como o opcode da instrução LUI é igual a 001111 (15 em binário), foi necessário fazer uma modificação na décima quinta posição da tabela (contando a partir de 0). Esta modificação foi feita a partir dos sinais de controle, pensando em quais deveriam estar ativados e quais não deveriam estar ativados. A figura abaixo mostra a ordem em que os sinais de controle estão organizados, sendo SHL16_Immed o sinal de controle referente ao bit mais significativo e RegDst o sinal de controle referente ao bit menos significativo:

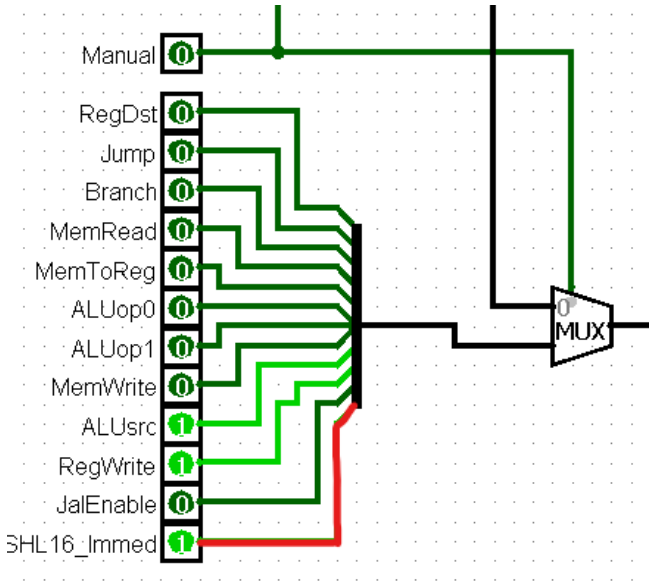



Figura 29: sinais de controle do processador MIPS monociclo.

Assim, como os sinais de controle SHL16_Immed , RegWrite e ALUSrc devem estar em 1, o número em binário que codifica esses sinais de controle será 101100000000, que é igual a b00 em hexadecimal. Portanto, adicionamos o número b00 na décima quinta posição da tabela da ROM:

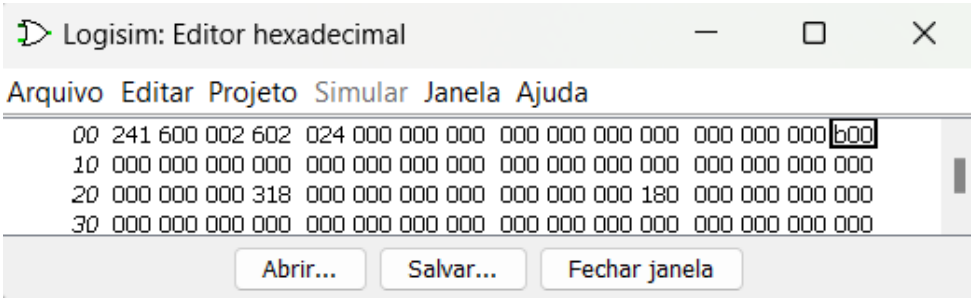



Figura 30: ROM do processador MIPS monociclo modificada.

## B) Implementação no Multiciclo

Para o Multiciclo, as adições no hardware foram idênticas às adições no monociclo, sendo adicionados novamente, um deslocador e um multiplexador com as mesmas finalidades. A principal diferença é que no valor de ALUSrc foi dividido em dois novos sinais de controle no valor de ALUSrcA (1 bit) e no valor de ALUSrcB (2 bits). Assim, como o valor 'shl16('immediate' se encontra na terceira entrada do segundo mux conectado à segunda entrada da ULA, e o valor de rs ($0) é repassado à primeira entrada da ULA, a soma dos dois resulta no valor 'Upper Immediate'.

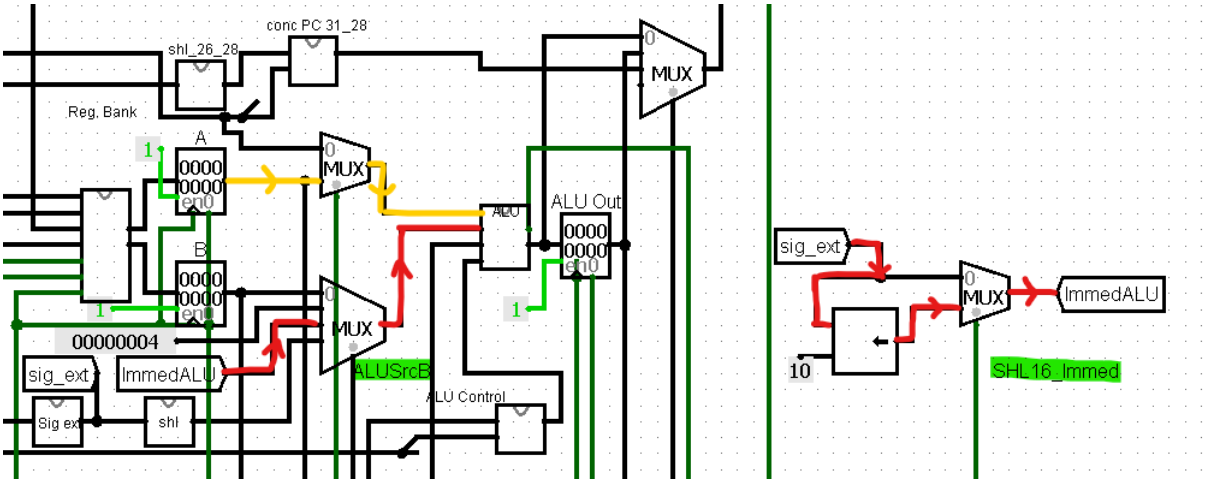



Figura 31: caminho que escolhe shl16('immediate') como segunda entrada da ULA, destacado em vermelho, com os sinais de controle destacados em verde e o caminho do valor de rt, destacado em amarelo.

- Novo sinal de controle SHL16_Immed

Para o multiciclo, também foi criado um novo sinal de controle SHL16_Immed, que é enviado para o multiplexador adicionado ao circuito. É evidente que para essa adição, é necessário fazer alterações nos distribuidores do circuito.

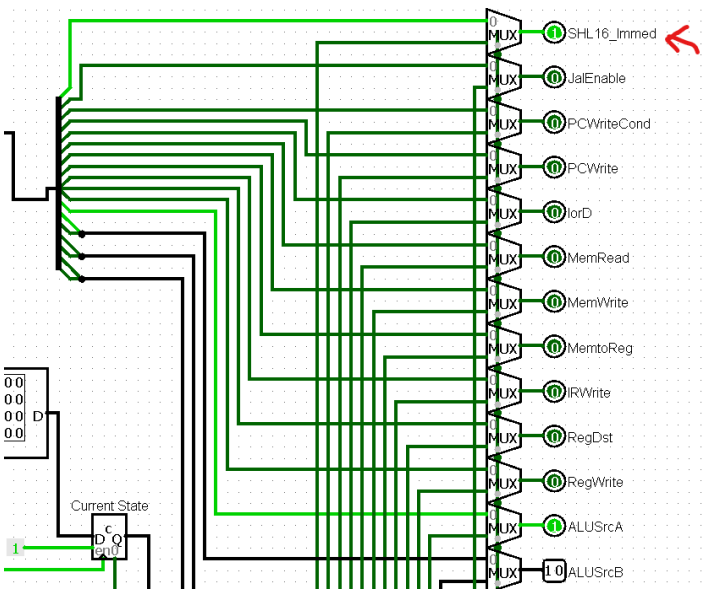


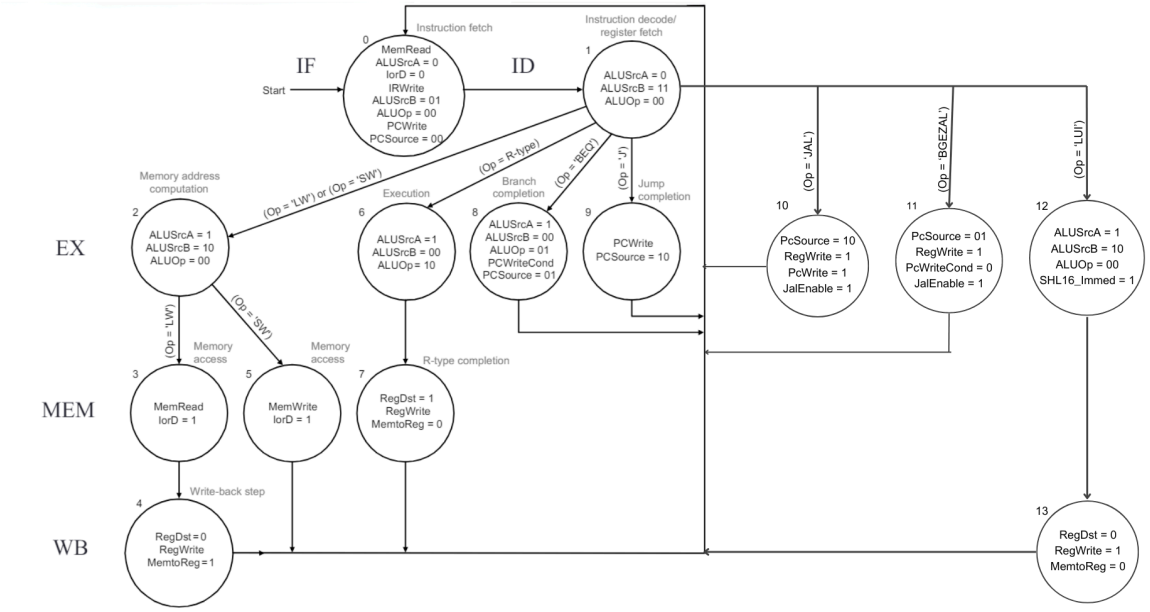
Figura 32: Imagem de dentro do bloco FSM_ROM. Novo sinal SHL16_Immed destacado pela setas vermelha.

- Modificação na ROM

Na tabela da ROM para o processador multiciclo, cada coluna representa um estado diferente. Dessa forma, foram criados dois novos estados: 12 e 13.

No estado 12, os seguintes sinais são diferentes de 0: AluSrcA (deve ser igual a 1), AluSrcB (deve ser igual a 10) e SHL16_Immed (deve ser igual a 1). Já no estado 13, RegWrite deve ser igual a 1.

Codificando para binário, temos o número 10 0000 0000 0110 0000, que é igual a (20060 em hexadecimal) na posição 12 da tabela, e 1000 0000 (80 em hexadecimal), na posição 13.


- Mudança na lógica de próximo estado

Na tabela da lógica de próximo estado, cada linha representa instruções e cada coluna representa qual é o próximo estado da determinada instrução. Como a instrução LUI tem opcode 001111, a linha na tabela que representa esta instrução é a linha de número 0f0. Assim, nesta linha colocamos os estados 1, 12 (“c” em hexadecimal) e 13 (“d” em hexadecimal), indicando os estados pelo qual a instrução deve passar durante a sua execução (lembrando que todas as instruções passam pelo estado 0 anteriormente).

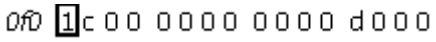



Figura 33: Lógica de próximo estado da instrução LUI

## C) Implementação no Pipeline

A implementação da instrução LUI no Pipeline foi realizada de forma quase idêntica às implementações no Monociclo e Multiciclo. A mesma lógica utilizada no Monociclo e no Multiciclo para repassar shl16(‘immediate’) para a ULA foi usada nessa implementação.

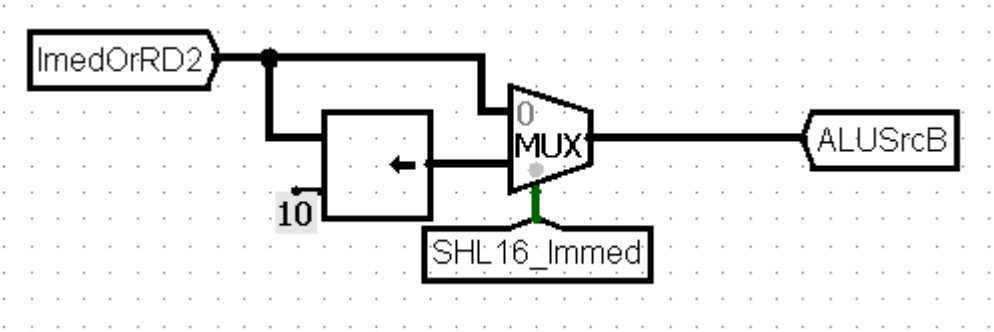



Figura 34: Hardware adicionado na versão pipeline.

Para isso, no terceiro estágio do pipeline, o imediato com sinal estendido (obtido no segundo estágio) será “shiftado” bits para a esquerda 16 e somado na ULA com $0. Nessa etapa também, RegDst terá o valor 0, pois no quinto estágio iremos escrever em rt.

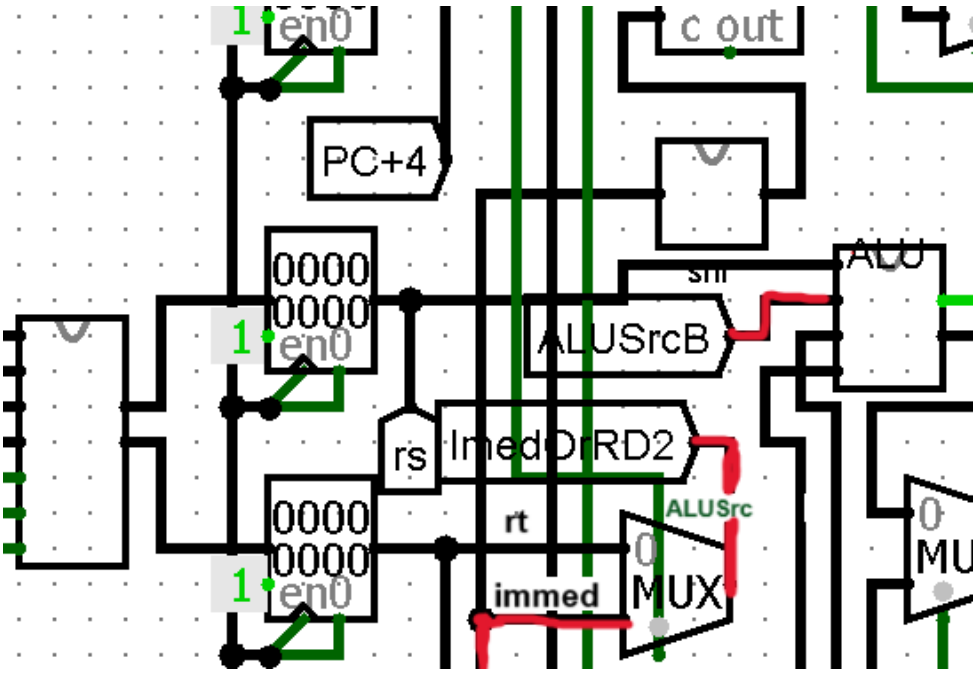



Figura 35: “Túneis” utilizados e seu caminho durante o terceiro estágio, destacado em vermelho.

No quarto estágio não há nenhum tipo de acesso a memória ou desvio. Por fim, no último estágio, ocorre a escrita do valor advindo da ULA (MemToReg == 0) em rt (RegWrite == 1).

# 4. Instrução BGEZAL (Branch On Greater Than Or Equal To Zero And Link)

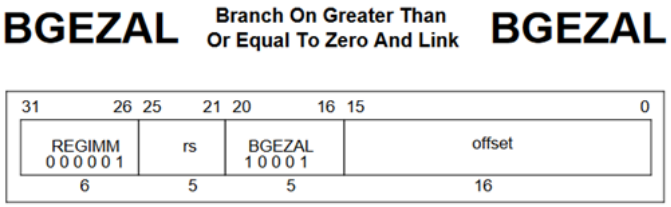



Figura 36: Formato da instrução BGEZAL

A instrução BGEZAL divide o *opcode* “000001” com as instruções BGEZ, BGEZALL, BGEZL, BLTZ, BLTZAL, BLTZALL e BLTZL, sendo diferenciada pelos bits 20-16. Entretanto, como esta é a única instrução com *opcode* “000001” presente em nossas implementações, optamos por utilizar somente seu opcode para representá-la, ignorando os bits 20-16.

## A) Implementação no Monociclo

A instrução BGEZAL, assim como a instrução JAL, armazena incondicionalmente o valor do Contador de Programa (PC) adicionado de 4 unidades no registrador 31 (*link register*). Portanto, aproveitamos partes do hardware introduzidas na implementação da instrução JAL, como o multiplexador que permite selecionar o valor 0x1f (representando o registrador 31) como opção de Write Address no banco de registradores e o sinal JalEnable, que é responsável por selecionar o registrador 31 como Write Address e por habilitar o valor de PC + 4 como dado a ser escrito neste registrador.

Foi adicionada uma lógica que verifica o bit mais significativo do valor armazenado no registrador rs e o sinal JalEnable, permitindo que um desvio seja realizado quando estes bits forem, respectivamente, iguais a 0 e 1; ou seja, durante a execução de uma instrução BGEZAL, será realizado um desvio se e somente se o conteúdo do registrador rs for um valor maior ou igual a zero.

O circuito responsável por implementar essa lógica está sendo representado em vermelho na figura abaixo.

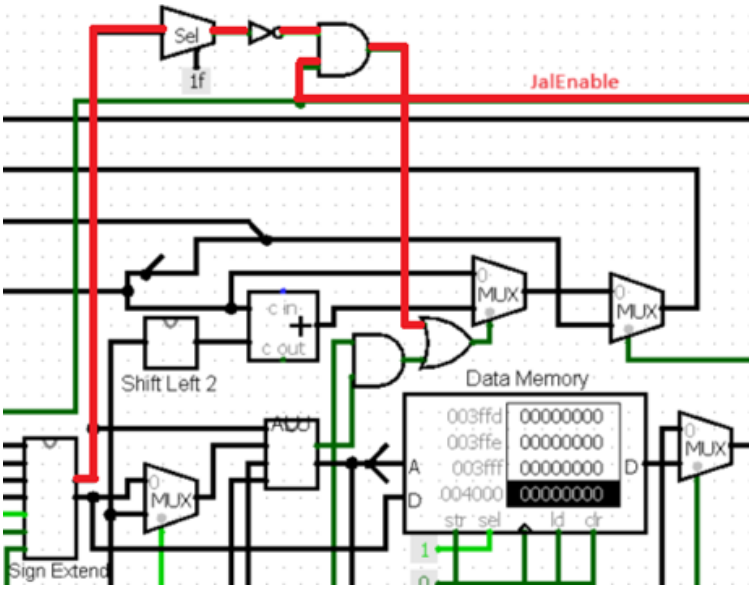



Figura 37: Implementação da lógica para verificar as condições do desvio para instrução BGEZAL no MIPS-Monociclo

Essa implementação, portanto, não utiliza a ALU para verificar se o conteúdo do registrador rs satisfaz as condições necessárias para realizar o desvio (diferentemente da instrução BEQ, que utiliza o sinal 'zero' da ALU). Além disso, essa implementação também não utiliza os sinais 'Jump' e 'Branch', pois o sinal 'JalEnable' e o inverso do bit mais significativo do conteúdo do registrador rs são suficientes para determinar se o desvio deve ou não ser realizado.

## B) Implementação no Multiciclo

A implementação da instrução BGEZAL no Multiciclo foi realizada de forma semelhante à implementação no Monociclo. Foi utilizada aqui a mesma lógica usada no Monociclo para verificar as condições de desvio: o desvio é realizado durante a execução da instrução BGEZAL se e somente se o inverso do bit mais significativo do registrador rs e o sinal 'JalEnable' estão ambos ativos, sem a necessidade de utilizar a ALU. Assim como no Monociclo, os sinais 'PcWriteCond' (análogo ao sinal 'Branch' do Monociclo) e 'PcWrite' (análogo ao sinal 'Jump' do Monociclo) não são utilizados para determinar se o desvio deve ou não ser realizado e, portanto, permanecem desativados.

Assim como no Monociclo, a lógica utilizada pela instrução JAL para selecionar o registrador 31 como Write Address no banco de registradores e escrever o valor PC + 4 neste registrador foi reaproveitada.

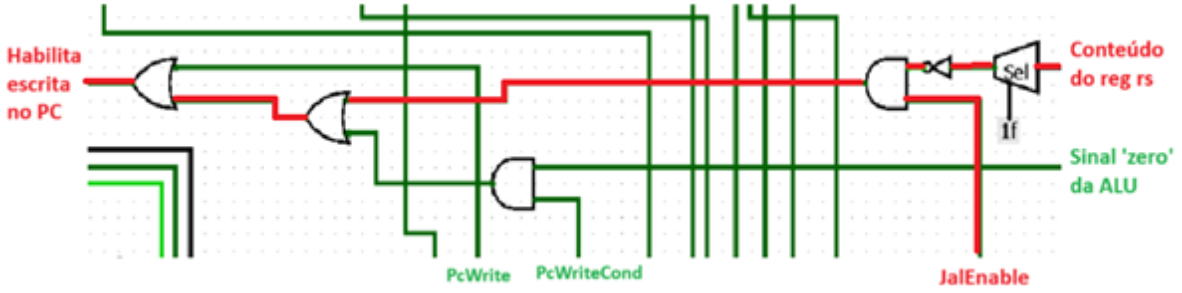



Figura 38: Implementação da lógica para verificar as condições do desvio para instrução BGEZAL no MIPS-Multiciclo

- Modificações na ROM e no diagrama de estados: Para suportar essa nova instrução, foi adicionado um novo estado à FSM (estado 11). Nesse estado, temos PcSource = 01 (endereço de destino do desvio), RegWrite = 1 (para escrever PC + 4 no registrador 31) e JalEnable = 1 (para selecionar o registrador 31 como Write Address e para possibilitar a realização do desvio). Note que, diferentemente de outras instruções que realizam desvio, temos PcWrite = PcWriteCond = 0, visto que nessa implementação estamos utilizando o JalEnable para permitir a realização do desvio.

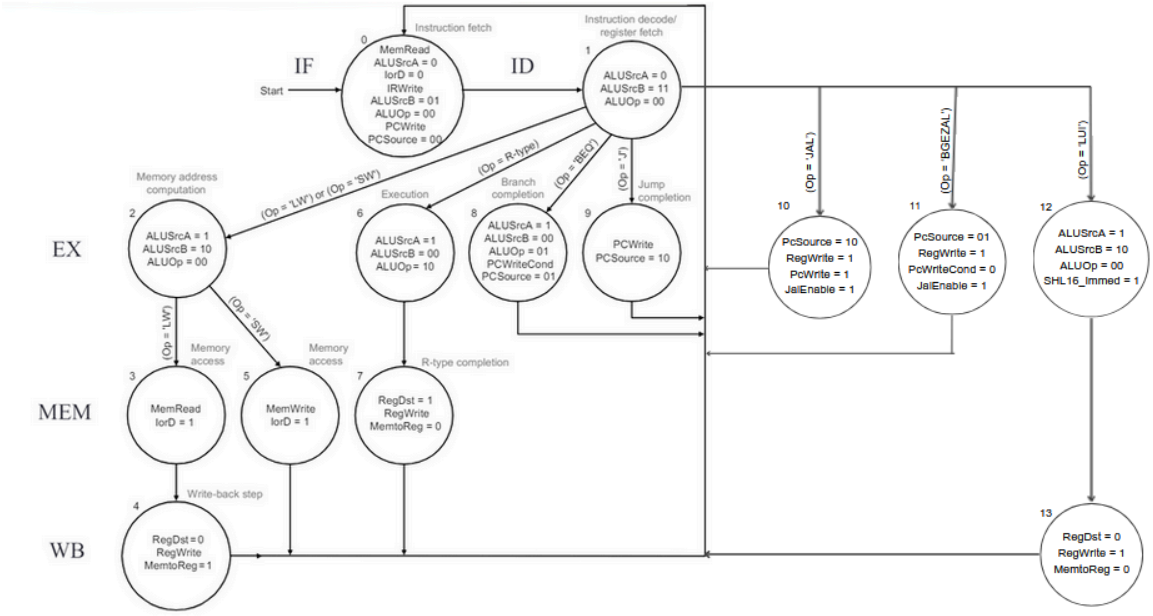



Figura 39: FSM com novo estado

```
0 05210 00030 00060 03000
4 00480 02800 00048 00180
8 18045 04002 14082 100c5
c 20060 00080 00000 00000
```

Figura 40: Tabela da ROM com os sinais de controle da instrução BGEZAL (*opcode* 000010)

```
010 1b00 0000 0000 0000
```

Figura x: Lógica de próximo estado da instrução BGEZAL

## C) Implementação no Pipeline

A implementação da instrução BGEZAL no Pipeline foi realizada de forma quase idêntica às implementações no Monociclo e Multiciclo. A mesma lógica utilizada no Monociclo e no Multiciclo para verificar as condições do desvio foi usada nessa implementação e o mesmo hardware adicional usado pela instrução JAL para selecionar o registrador 31 como Write Address e por habilitar o valor de PC + 4 como dado a ser escrito neste registrador foi reutilizado. Assim como nas implementações no Monociclo e Multiciclo, não estamos utilizando a ALU nesta instrução.

Para realizar o desvio, armazenamos, no terceiro ciclo do pipeline, o valor resultante da conjunção do inverso do bit mais significativo do valor do registrador rs com o sinal JalEnable em um registrador. No quarto ciclo, lemos o valor armazenado neste registrador e, caso este seja igual a 1, atualizamos o PC com o endereço do desvio.

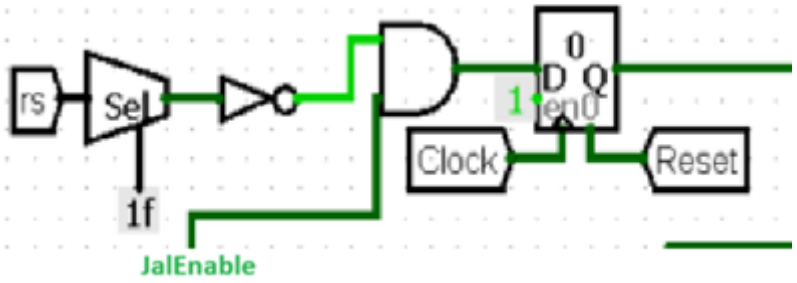



Figura 41: Implementação da lógica para verificar as condições do desvio para instrução BGEZAL no MIPS-Pipeline