Organização de Computadores B (2022/1) - Turma A

Trabalho 1 - Implementação de Instruções MIPS

Grupo: 3

Henrique Utzig - 00319043

João P. Cosme - 00314792

Matheus Stein de Aguiar - 00302320

Victor Machado Gonçalves - 00313930

Descrição das Instruções:

ADDI – Add Immediate

TIPO I

ADDI 0 0 1 0 0 0	rs	rt	immediate
6	5	5	16

A operação é simples. Queremos ler registro rs a partir do banco de registradores e somar com o valor imediato que é estendido com sinal para 32 bits. A soma em si é efetuada por um ALU. Simplificadamente:

$$\text{GPR[rt]} \leftarrow \text{GPR[rs]} + (\text{immediate}_{15})^{16} || \ \text{immediate}_{15\ldots0}$$

Para deixar mais claro a notação, irei explicar alguns trechos:

GPR[ra]	Ler o registro ra do banco de registradores
$\text{abc}_{x\ldots y}$	Obter os bits de x a y(inclusive) da variável abc, podemos omitir y, levando somente a 1 bit.
$(abc)^x$	Copiar sequência de bits da variável abc e unir sequencialmente x vezes
$abc \ \| \ xyz$	Unir sequência de bits abc e xyz, preservando a ordem dada

BLTZAL – Branch On Less Than Zero And Link

TIPO I

REGIMM 0 0 0 0 0 1	rs	BLTZAL 1 0 0 0 0	offset
6	5	5	16

Temos uma instrução Branch, onde uma decisão deve ser tomada para que se saiba qual a próxima instrução a ser executada. Funcionalmente, iremos pular para PC+offset caso o valor armazenado no registrador rs seja menor que 0. Para isso, podemos simplesmente utilizar o bit 31(considerando que os bits de um registrador de uso geral no MIPS são numerados de 31 a 0) do valor armazenado no registrador rs. Quando trabalhamos com valores que podem assumir valores positivos ou negativos, o bit mais significativo é igual a 1 quando o número representado é negativo, ou seja, é menor que zero.

Seguindo uma ordem aproximada queremos:

1. Offset é sempre de 4 em 4 bits, portanto, não é necessário incluir os dois primeiros zeros. Assim, queremos incluir dois zeros no início (equiv. a multiplicar por 4). Temos 18 bits.
2. Estender por sinal o offset por 14 bits levando a $18\ \text{bits} + 14\ \text{bits} = 32\ \text{bits}$
3. Com o tamanho adequado, somar com o PC.

A partir daqui, vemos as diferenças com a instrução BEQ com a qual estamos acostumados. Primeiramente, um link é feito com a próxima instrução a ser realizada caso não haja *branch,* para isso, o registrador 31 (registrador de retorno) é utilizado. Além disso, ao invés de utilizarmos a ULA para realizar uma comparação, apenas testamos um único bit do registrador rs carregado.

Desta forma, temos os passos finais abaixo:

1. Armazenar PC+8 no registrador $\text{GPR}[31]$
2. Verificar se o bit 31 de rs está ativado, se sim, pular

Esse funcionamento se dá igual em todas as implementações do MIPS, com o cuidado de que no Pipeline o valor armazenado é PC+8 ao invés de PC+4, pois há a presença do *delayed slot* onde uma instrução não relacionada ao *branch* pode ser executada para que não haja uma perca de ciclos no caso de um *branch* diferente do previsto.

```
target      ← (offset_15)^14 || offset || 0^2
condition   ← (GPR[rs]_31 = 1)
if condition then
   PC       ← PC + target
endif
```

MULT – Multiply

TIPO R

SPECIAL 0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0	MULT 0 1 1 0 0 0
6	5	5	10	6

A primeira operação R. De alto nível, seu funcionamento é bem simples, não é necessária uma explicação, só devemos reparar que, como estamos em 32 bits, o valor retornado pela multiplicação é de 64 bits. Como o nosso MIPS é de 32 bits, devemos criar dois registradores LO e HI, como proposto no manual.

```
t   ← GPR[rs] * GPR[rt]
LO  ← t31...0
HI  ← t63...32
```

JR – Jump Register

TIPO R

SPECIAL 0 0 0 0 0 0	rs	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	JR 0 0 1 0 0 0
6	5	15	6

Por último, temos uma instrução de pulo. Repare que a instrução é do tipo R. Temos uma pista que podemos utilizar o ALU de alguma forma para efetuar a operação. OBS: somar com zero e corrigir bits de controle

```
temp ← GPR[rs](+ 0)
PC   ← temp
```

MIPS – Monociclo

ADDI

Modificações Estruturais

Não é necessário realizar nenhuma modificação estrutural.

Modificações no Controle

Sinal	Valor
AluSrc	1
RegWrite	1

Comentários

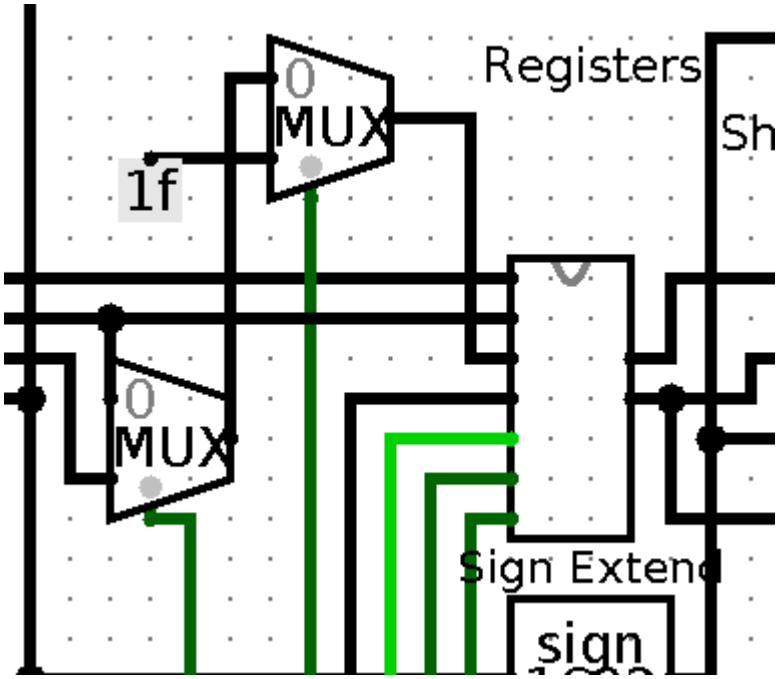
A instrução ADDI é equivalente ao código 003 do bloco de comando, onde as únicas flags ativas são o AluSrc(precisa controlar a ULA pelo OpCode e não pelo funct visto que não é uma instrução R) e o RegWrite (precisa escrever no registrador), visto que não mexe na memória (MemWrite, MemRead), não tem pulo (JMP, Branch), o registrador destino é o rt (RegDst = 0), a operação realizada pela ULA é a soma (AluOp1 = 0, AluOp2 = 0) e como iremos usar o valor que vem da ULA e não da memória (MemToReg = 0).

BLTZAL

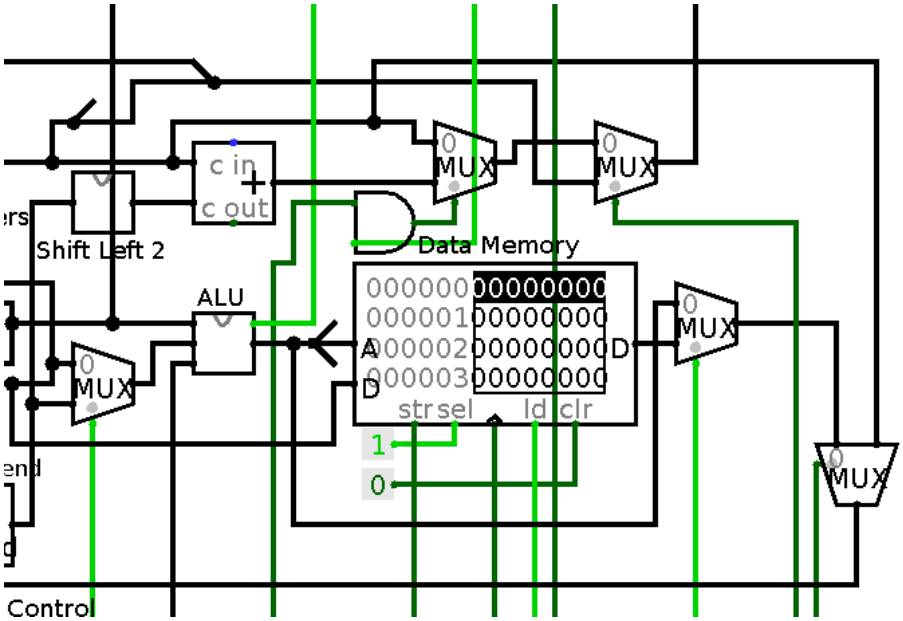
3 ações devem ser tomadas para a implementação do BLTZAL no Monociclo:

- Writeback no registrador 31 do valor de PC+4
- Comparação de rs[31] com 1 para verificar se rs é menor que zero
- Escolha entre o sinal de zero da ULA e o sinal emitido pela nova comparação

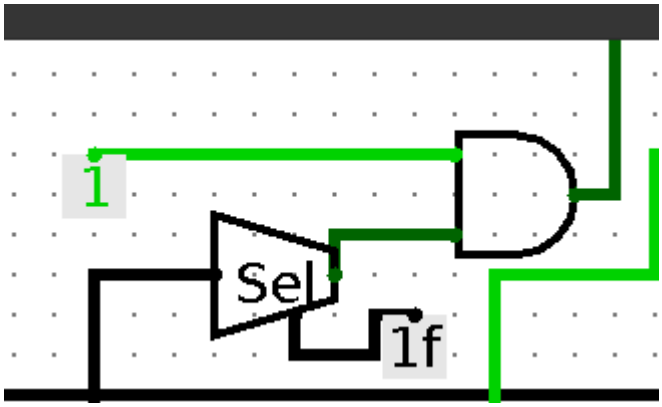
Para a primeira ação, utilizamos uma constante 31 e um novo mux na porta do banco de registradores. Este MUX não impacta diretamente no caminho critico já que, por estar no final do caminho, tem todo o ciclo para se estabilizar e não deve acarretar em um tempo maior de período necessário. Este MUX por padrão não altera o caminho atual, mas quando ativado pelo novo bit de controle “Link”, escolhe pelo valor de PC+4+offset:



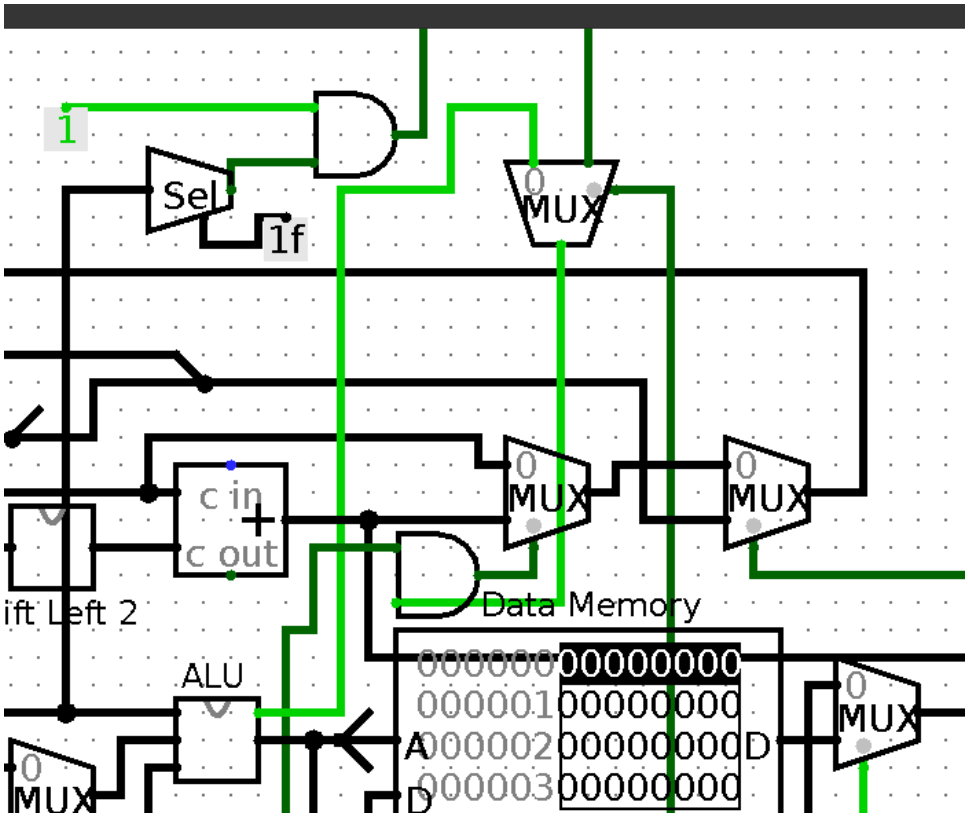
Para o envio do endereço de PC+4+offset, ainda devemos enviar o resultado de PC+4 para o banco de registradores. Para isso, um novo MUX após a saída do banco de dados foi adicionado. Este MUX também é controlado por “Link” e, já que se incorpora ao caminho de LW, pode impactar negativamente o período do circuito. Uma alternativa seria o incremento do MUX já existente para um 3-para-1, porém não foi a implementação escolhida.



Em seguida, para a comparação entre rs[31] e o bit de controle “Link”, um simples AND em paralelo com a ULA pode ser adicionado. Esta alteração garante que o bit será apenas 1 no caso tanto do bit de controle estar ativo quanto rs ser um número negativo. Por estar em paralelo com a ULA, não há alteração no caminho crítico.



Por fim, um novo MUX é adicionado entre a saída “Zero” da ULA e a saída do novo comparador, isto nos garante que o bit enviado para ser comparado com o bit “branch” é o correto e não interferimos no funcionamento do BEQ no caso do valor do rs ser menor que zero. O tempo adicional deste MUX pode ser considerado pois não faz parte do caminho crítico do LW, logo, não impacta no período de tempo necessário para que ele seja realizado.



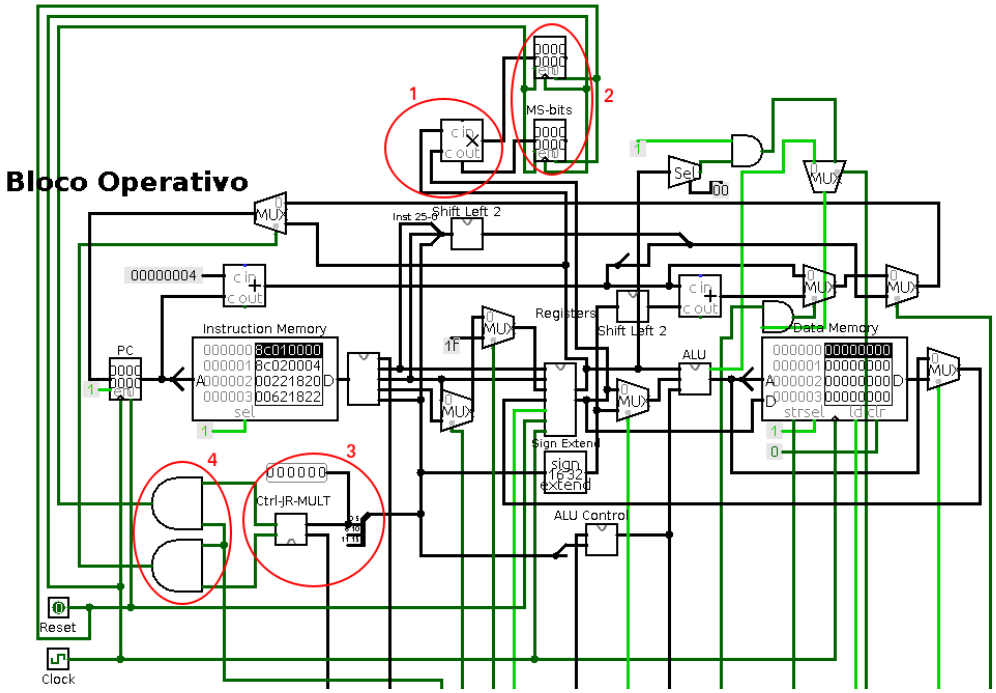
Os bits de controle, por fim, são idênticos ao de BEQ, com a diferença de que o bit de controle “Link” deve ser 1.

Assim, os bits de controle são:

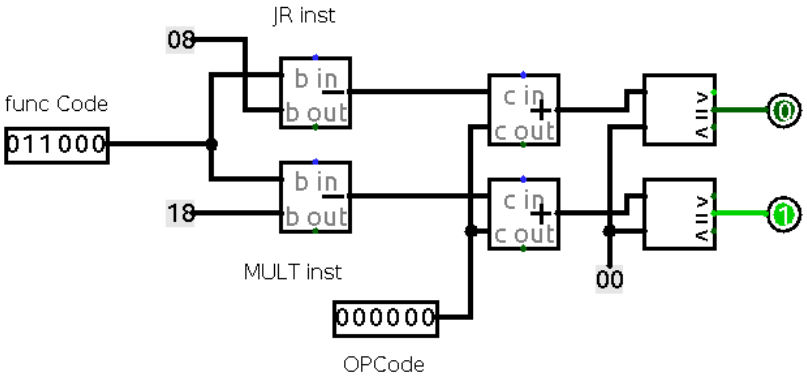
Instrução	Sinal “Link”
BLTZAL	1
BEQ	0
Demais	x

MULT

Para implementarmos a instrução MULT no MIPS monociclo precisamos adicionar os seguintes componentes:



1. Foi adicionado um bloco aritmético multiplicador, o qual recebe como entrada os valores dos registradores *Rs e Rt;*
2. Foram adicionados dois registradores que possuem como entrada as saídas do bloco multiplicador, o registrador mais ao topo recebe os *bits* menos significativos do resultado da multiplicação e o registrador logo abaixo recebe os *bits* mais significativos;
3. O controle da escrita do resultado de multiplicação é feito pelo bloco de controle *Ctrl-JR-MULT* (esse bloco é compartilhado com a instrução JR), o qual tem como entrada o campo *funct* e o *Opcode* da instrução e gera um sinal de controle que habilita a escrita dos registradores de *bits* mais significativos e menos significativos do resultado da multiplicação;



4. Por fim, foi adicionada uma porta *AND* em série a saída do bloco de controle criado, essa porta possuí como entrada o sinal de controle para a instrução

MULT do bloco criado e o novo sinal de controle *Special* criado para tratar os casos especiais das instruções *MULT e JR.*

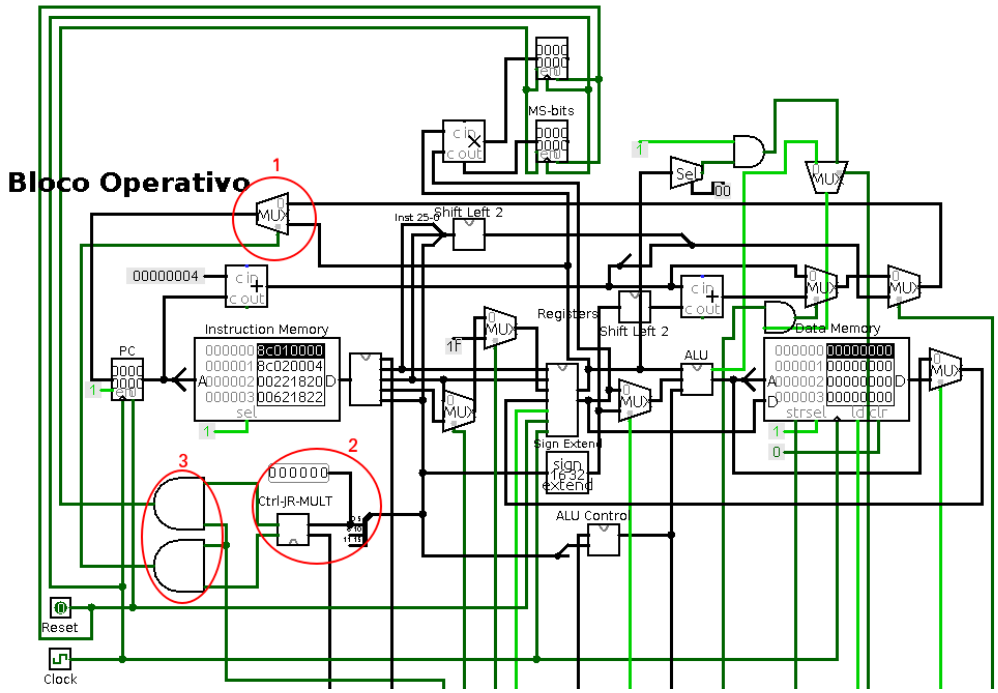
A instrução MULT funcionará com os seguintes sinais de controle:

RegDst	Jump	Branch	MemRead	MemToReg	ALUOp	MemWrite	ALUSrc	RegWrite	Special
X	0	0	X	X	XX	0	X	0	1

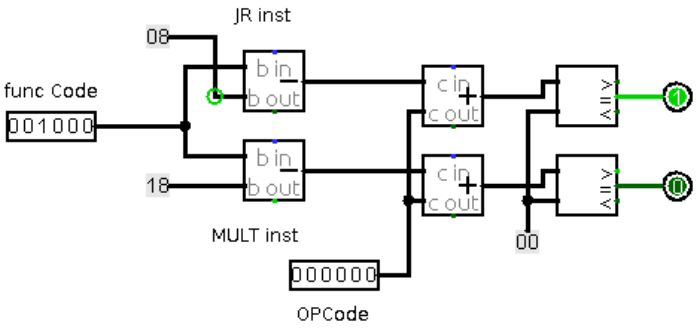
MULT não escrever em nenhum registrador presente no banco de registradores e nem na memória de dados então os sinais de escrita serão zerados (*RegWrite e MemWrite*). A instrução não realiza desvio, então os sinais *Jump e Branch* serão zerados, além disso, não utilizados a *ALU* para realizar a multiplicação então os sinais *ALUOp e ALUSrc* podem assumir qualquer valor e não irão alterar o resultado da multiplicação. Como não é utilizado o caminho de dados comum, os sinais dos multiplexadores *RegDst e MemToReg* podem assumir qualquer valor. Não vamos ler da memória, então o sinal *MemRead* também pode assumir qualquer valor. Por fim, devemos inserir o valor lógico 1 no novo sinal de controle criado *Special*, o qual trata das instruções MULT e JR.

JR

Para implementarmos essa instrução no MIPS monociclo os seguintes componentes foram adicionados:



1. Fui adicionado um multiplexador antes da entrada de dados do registrador *PC,* esse mux têm como entradas o antigo caminho dos dados de *PC* (saída do multiplexador que seleciona PC+4 ou PC+4+Label) e a segunda entrada está conectada diretamente à saída do registrador *Rs;*
2. Foi criado um bloco de controle para diferenciar as instruções JR e MULT, ambas do tipo R. Esse bloco verifica o *OPcode* e o campo *funct* da instrução e gera um sinal de controle para o multiplexador que realiza o salto JR;



3. Por fim foi adicionado uma porta lógica AND que controla o multiplexador (1), essa porta lógica tem como entradas a saída do bloco de controle (2) e a segunda entrada é o novo sinal de controle *Special* o qual foi adicionado para tratar as novas instruções JR e MULT.

A instrução JR necessita dos seguintes sinais de controle para funcionar:

RegDst	Jump	Branch	MemRead	MemToReg	ALUOp	MemWrite	ALUSrc	RegWrite	Special
X	0	0	X	X	XX	0	X	0	1

A instrução JR não escreve em nenhum registrador nem na memória então os sinais de escrita (*MemWrite* e *RegWrite*) são zerados. Não precisamos utilizar a *ALU* nessa instrução então os sinais que selecionam entradas e operações (*ALUSrc* e *ALUOp*), logo, esses sinais podem ter qualquer valor. Além disso, os seguintes sinais complementares da memória, *ALU* e banco de registradores podem receber qualquer valor: *RegDst, MemRead* e *MemToReg.* Por fim, zeramos os sinais *Branch e Jump* para evitarmos desvios indesejáveis, e o novo sinal *Special* é o único sinal de controle a necessariamente receber valor lógico 1.

MIPS – Multiciclo

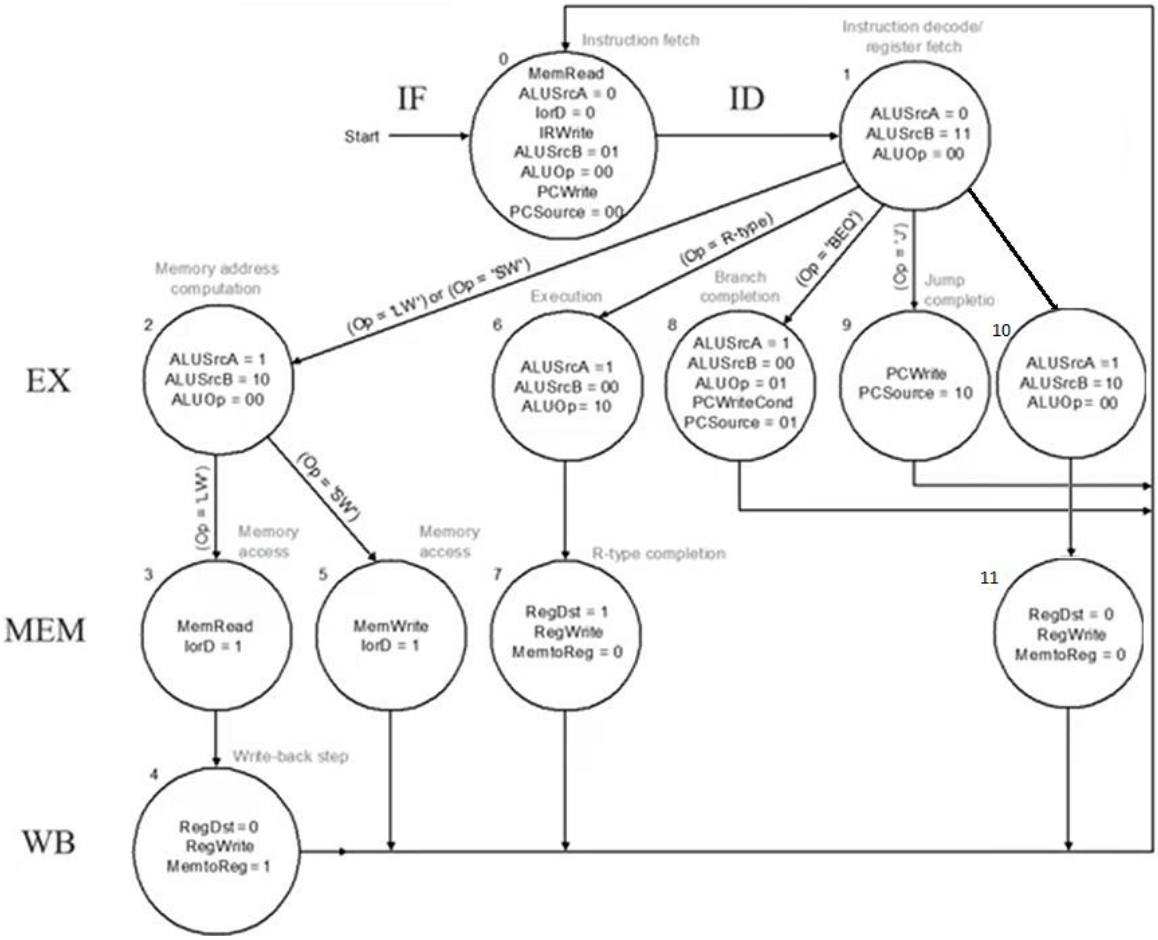
ADDI

Modificações Estruturais

Não é necessário adicionar nenhuma nova estrutura.

Modificações no Controle

É necessário adicionar novos estados a partir do estado 1, como pode ser visto na figura a seguir.

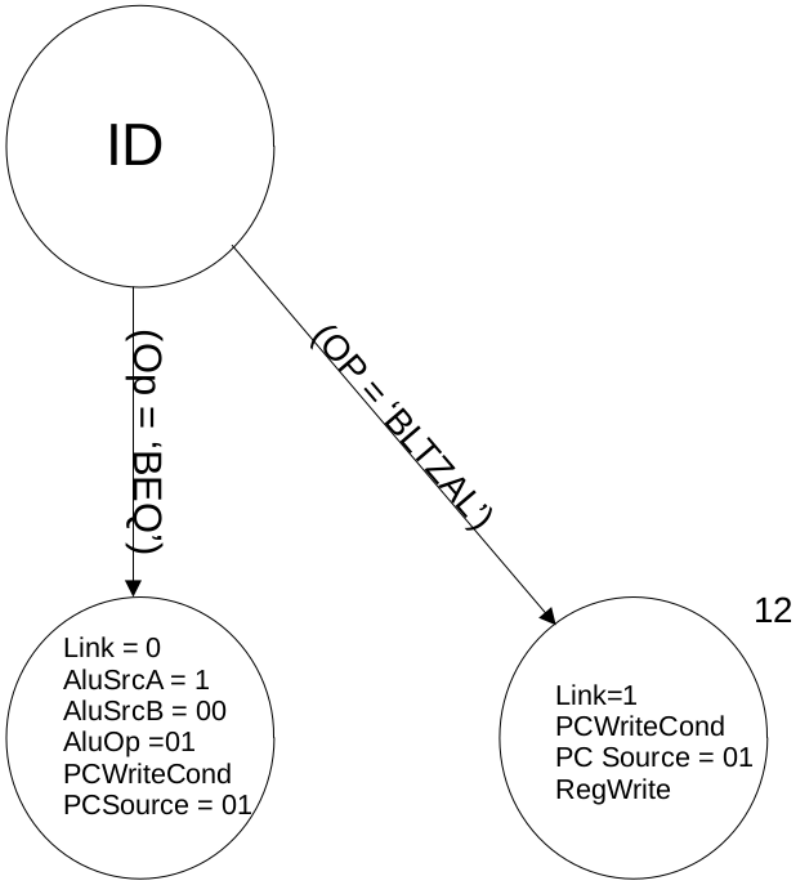


Comentários

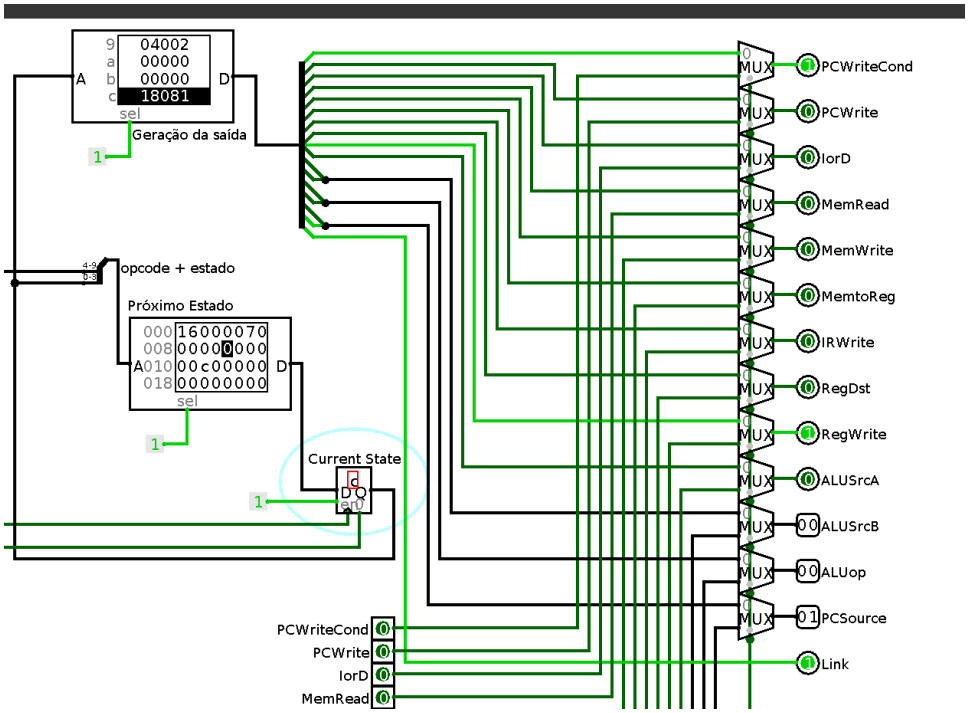
No estado 10 usaremos a ULA para fazer uma soma do registrador r com o imediato, por isso ALUSrcA = 1 (usar o regr), ALUSrcB = 2 (usar o imediato) e ALUOp = 00 (soma). E então no estado 11 guardaremos o valor da soma no registrador t, RegDst = 0 (seleciona para guardar no RegT), RegWrite = 1 (queremos escrever no registrador), MemtoReg = 0, (queremos receber do ALUOut não do MDR).

BLTZAL

Para a adição deste comendo, a seguinte alteração deve ser feita no estado 08, além da adição do novo estado 12 na parte de EX. Naturalmente, o estado 12 leva diretamente a IF.

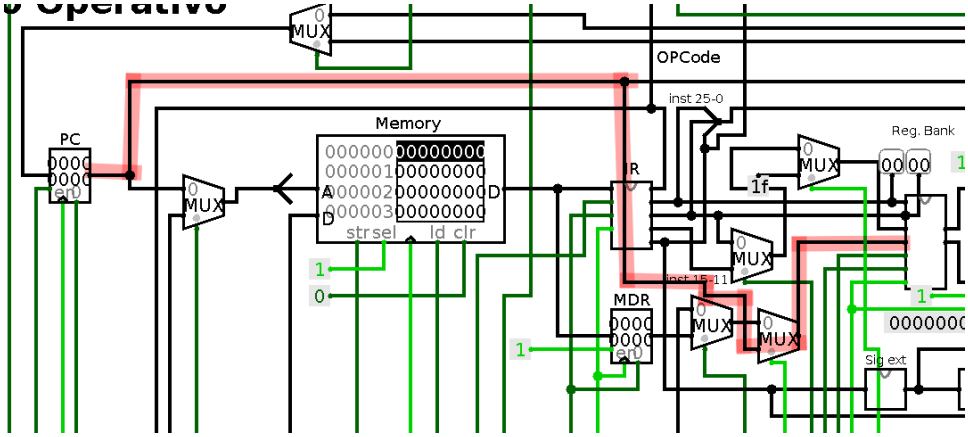


A alteração na FSM segue como abaixo:

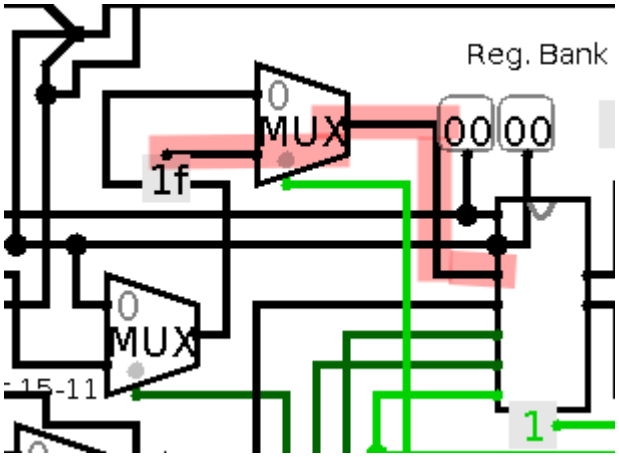


Já a implementação no bloco operativo, segue os mesmos moldes que a implementação do BLTZAL no Monociclo, porém tomando cuidado para não causar *side-effects.*

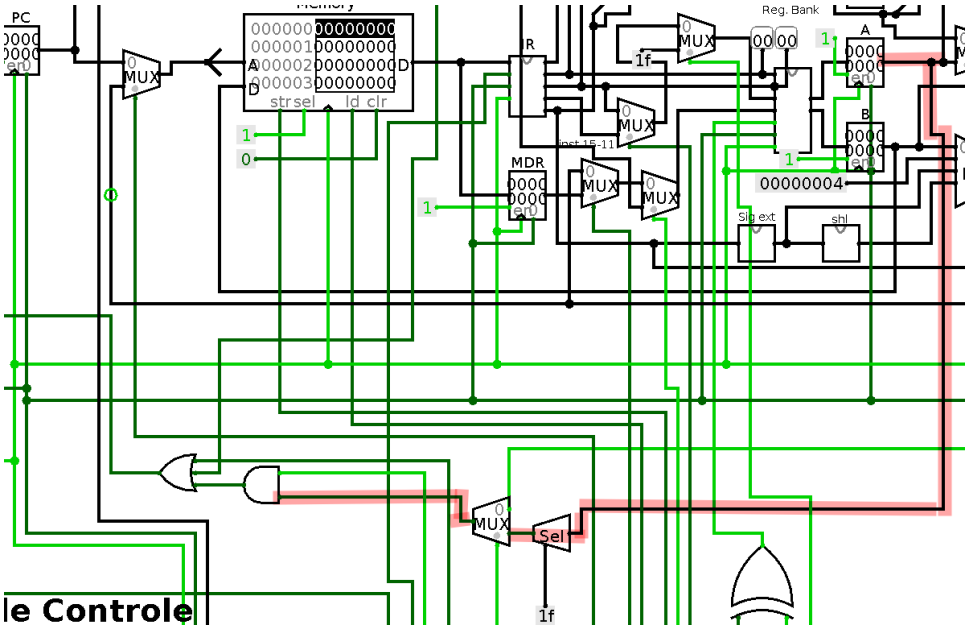
Primeiramente, devemos conseguir enviar PC+4 para a porta de escrita do banco de registradores, segue abaixo:



Além do valor de PC+4, devemos apontar o endereço 31 para escrita, segue abaixo:



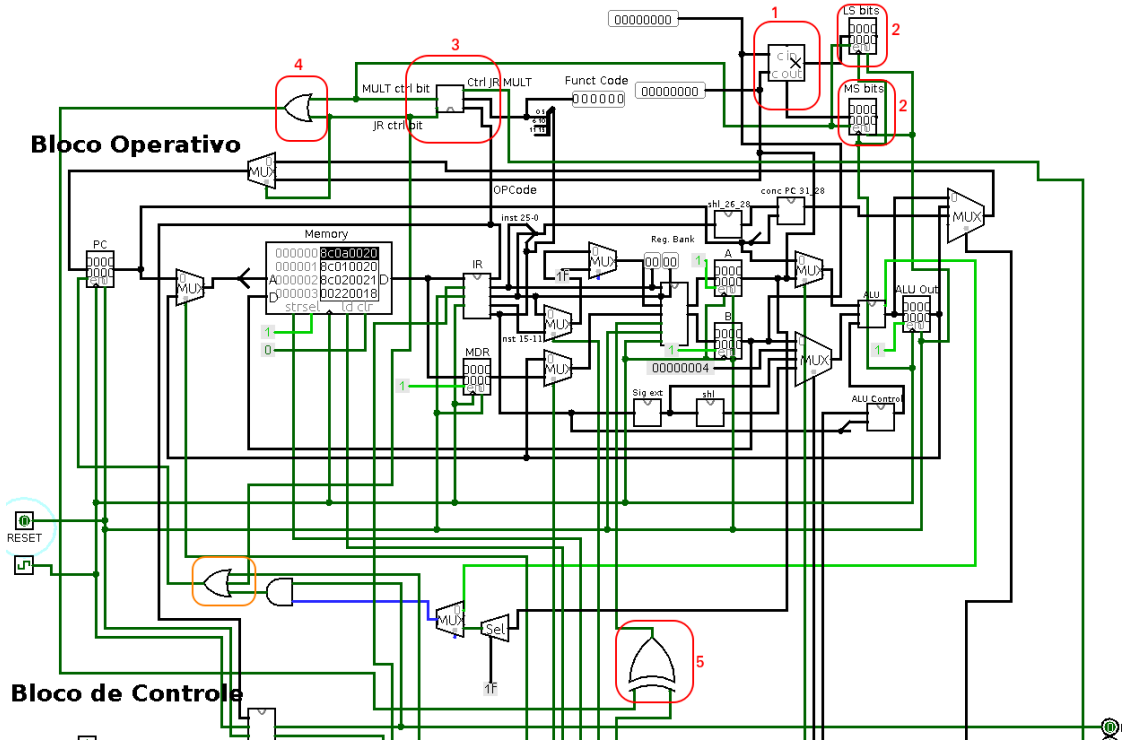
Já a decisão em si, onde testamos o bit 31 do registrador rs (no estado 12 armazenado em A, após o carregamento em ID), está representada aqui:



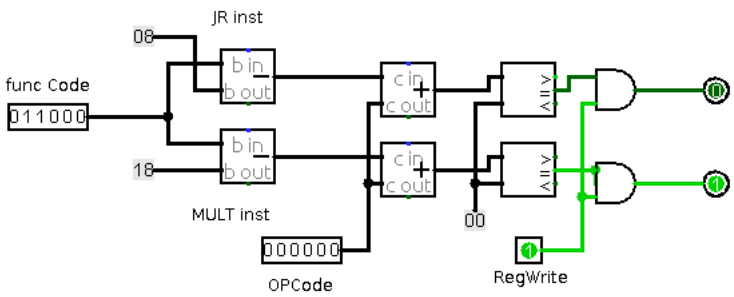
Todos os multiplexadores introduzidos são controlados pelo novo sinal de controle “Link”, assim como no Monociclo.

MULT

Para implementarmos a instrução MULT no MIPS multiciclo precisamos adicionar os seguintes componentes:



1. Foi adicionado um bloco aritmético multiplicador, o qual recebe como entrada os valores dos registradores Rs e Rt;
2. Foram adicionados dois registradores, que irão guardar o resultado da multiplicação, sendo o registrador mais ao topo o dos *bits* menos significativos e logo abaixo o dos *bits* mais significativos;
3. O controle de escrita do resultado na multiplicação nos registradores é feito pela saída "MULT ctrl bit" do bloco "Ctrl JR MULT", esse bloco é compartilhado com a instrução JR. Em suma, esse bloco possuí como entrada o campo *Opcode* e o campo *funct* da instrução e o sinal de controle *RegWrite* e a partir desses valores coloca a saída "MULT ctrl bit" em nível lógico alto caso tenhamos uma instrução MULT e o sinal *RegWrite* estiver ativo;



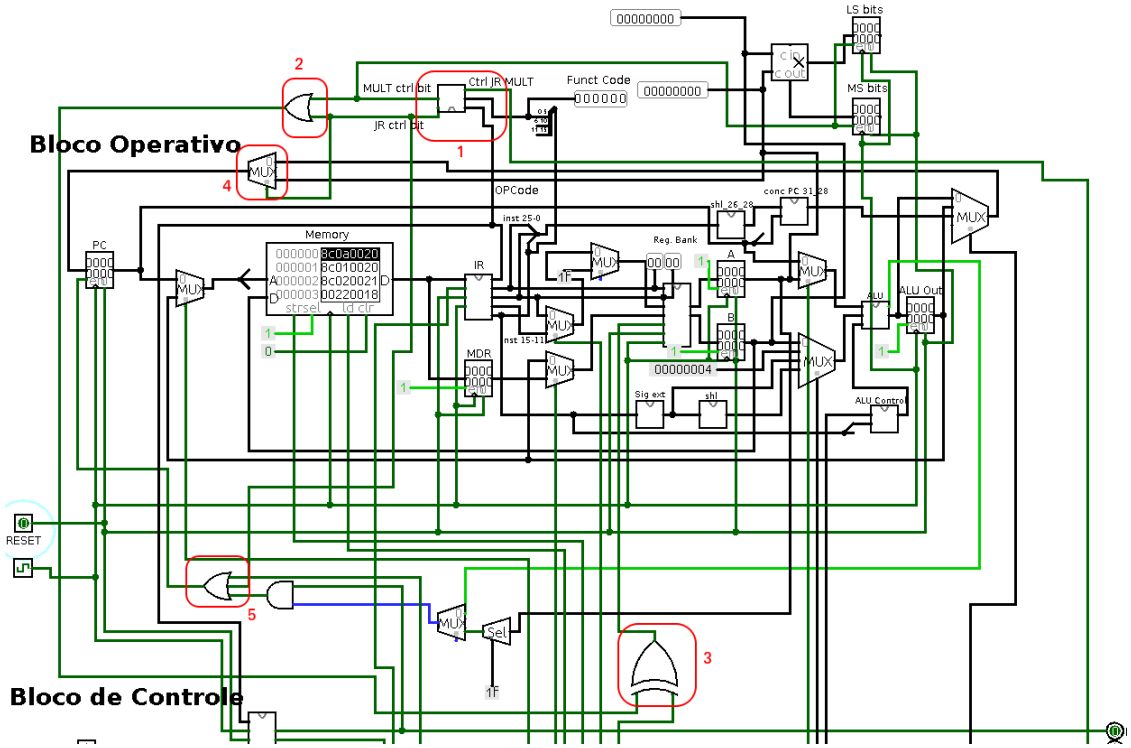
4. Como vamos usar o sinal de controle *RegWrite* para escrever nos registradores da multiplicação, devemos evitar que o valor presente no registrador da *ALU*

seja escrito no banco de registradores. Para evitarmos esse problema, foi adicionado uma porta lógica OR com as duas saídas do bloco criado como entradas e sua saída entra na porta lógica XOR (5), que a sua outra entra é o sinal *RegWrite* e sua saída entra no banco de registradores. Assim, quando tivermos uma instrução MULT ou JR não vamos escrever lixo no banco de registradores.

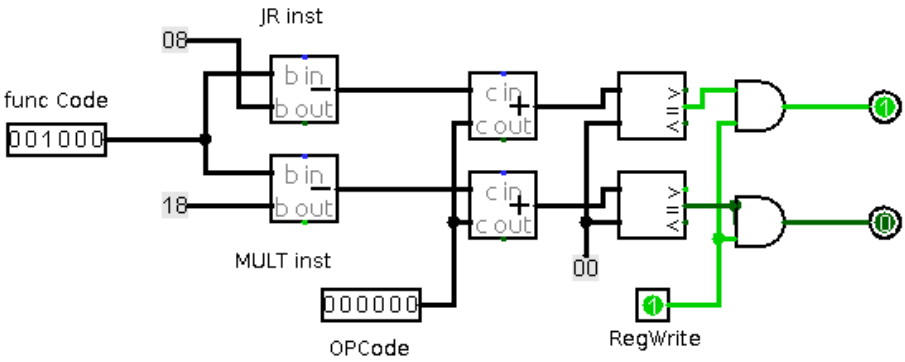
Como os sinais de controle dessa instrução são gerados pelo bloco criado, não foi criado um novo estado no controle original do processador. Assim, a instrução MULT segue o mesmo caminho de instruções do tipo R na máquina de estados que descreve o controle do MIPS multiciclo. Todavia, vale lembrar que mesmo seguindo o caminho de uma instrução do tipo R não escrevemos nenhum valor no banco de registradores.

JR

Para implementarmos a instrução JR no MIPS multiciclo precisamos adicionar os seguintes componentes:



1. Foi criado o bloco "Ctrl JR MULT" que gera os sinais de controle para as instruções JR e MULT. Esse bloco recebe como entradas os campos *Opcode e funct* da instrução, e também recebe o sinal de controle *RegWrite.* O bloco verifica se a instrução atual é uma instrução JR e se o sinal *RegWrite* está habilitado, caso esses dois casos forem verdadeiros o sinal de saída do bloco "*JR ctrl bit"* é acionado;



2. Foi adicionada a porta lógica OR (2), que tem como entrada ambos os sinais de saída do bloco "Ctrl JR MULT", essa porta lógica serve para caso qualquer um dos sinais de saída do bloco esteja habilitado a entrada de escrita no banco de registradores seja desabilitada, assim não será escrito lixo em algum registrador;

3. A porta lógica XOR (3) recebe como entrada o sinal *RegWrite* e a saída da porta lógica OR (2), e sua saída está conectado no pino que habilita a escrita no banco de registradores. Desse modo, desabilitamos a escrita no banco de registradores sempre que uma instrução JR ou MULT está sendo executada;
4. Foi adicionado o MUX (4) antes da entrada de dados do registrador PC, esse mux tem como entradas os valores do registrador A e a saída do multiplexador controlado pelo sinal de controle *PCSource,* ainda esse mux é controlado pela saída "JR ctrl bit" do bloco criado. Assim, sempre que tivermos uma instrução JR vamos conectar a entrada de dados do registrador PC com a saída do registrador A (esse registrador tem o valor do registrador *Rs*);
5. Por fim, para habilitarmos a escrita do valor do registrador no PC, expandimos a porta lógica OR que controlava a escrita do registrador PC e a nova entrada é conectada ao sinal da saída "JR ctrl bit" do bloco "Ctrl JR MULT" criado.

Como os sinais de controle dessa instrução são gerados pelo bloco criado, não foi criado um novo estado no controle original do processador. Assim, a instrução JR segue o mesmo caminho de instruções do tipo R na máquina de estados que descreve o controle do MIPS multiciclo. Todavia, vale lembrar que mesmo seguindo o caminho de uma instrução do tipo R não escrevemos nenhum valor no banco de registradores.

MIPS – Pipeline

ADDI

Modificações Estruturais

Só é necessário configurar o bloco de controle para que essa instrução seja possível.

Modificações no Controle

Jump, Branch, MemRead E MemWrite são 0 pois não usamos, RegDst=0 (queremos gravar no RegT), ALUOp = 00 (soma), ALUSrc = 1 (queremos o imediato), MemToReg = 0 (queremos o valor que vem direto da ULA, não irá ler nada da memória), RegWrite = 1 (irá escrever no registrador).

Sinal	Valor
Jump	0
Branch	0
MemRead	0
RegDst	0
ALUOp	00
ALUSrc	1
MemToReg	0
RegWrite	1

BLTZA

Modificações Estruturais

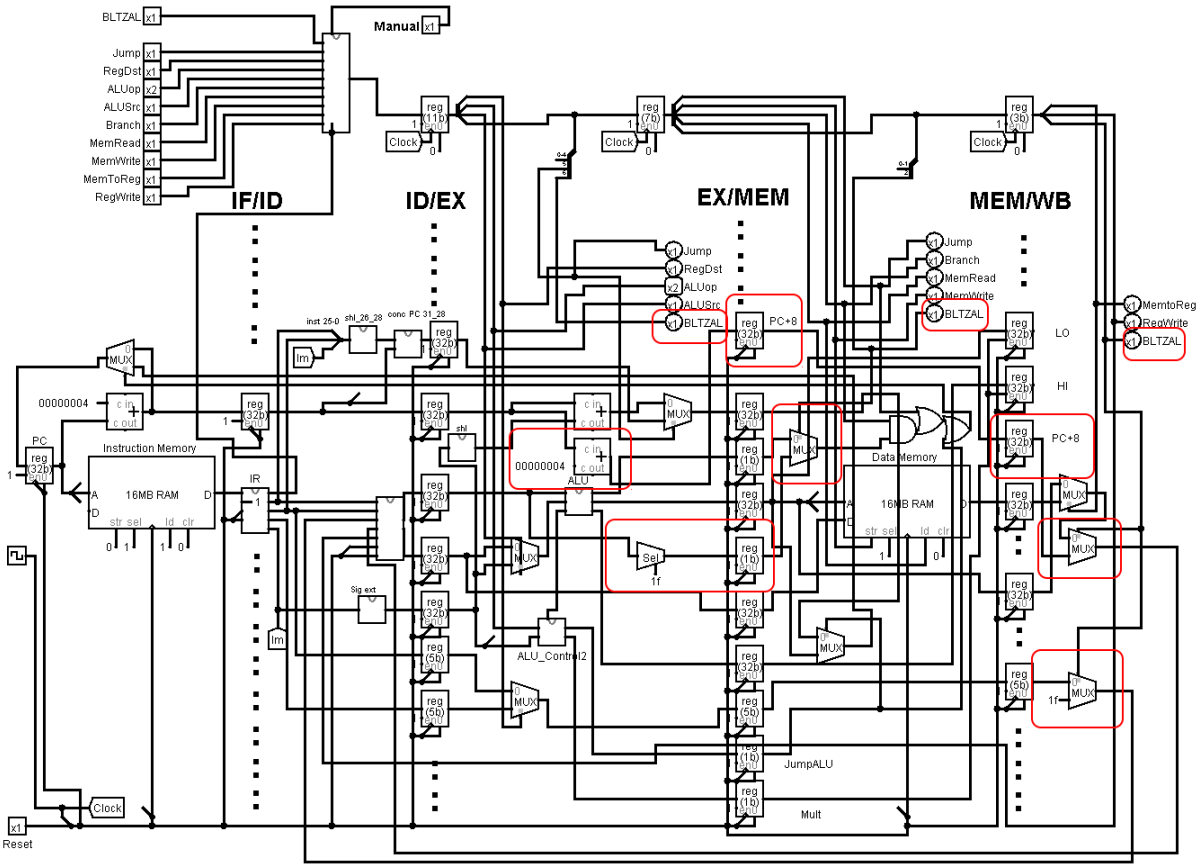
Foi incluído um somador para efetuar (PC+4) + 4 = PC+8(pular delayed slot) e dois registradores para armazenar o resultado (um em EX/MEM e outro em MEM/WB).

Foi incluído um selecionador de bit (maior bit significante, bit de sinal) e um registrador de 1 bit para armazenar durante o pipeline.

Foi incluído três multiplexadores controlados pelo bit de controle BLTZAL:

Um multiplexador controla o tipo de condição devemos verificar para o branch. Igualdade (BEQ) ou verificar se é negativo (BLTZAL).

Depois temos dois multiplexadores, um controla o que deve ser armazenado no registrador na hora de WB. No caso, armazenar o PC+8 ou outro valor fora disso. Por último, o mesmo bit de controle também escolhe qual registrador deve ser armazenado o valor, ou seja, para BLTZAL, o registrador 31.



Modificações no Controle

Em termos de bit de controle, temos o seguinte:

Sinal	Valor
Branch	1
BLTZAL	1
RegWrite	1

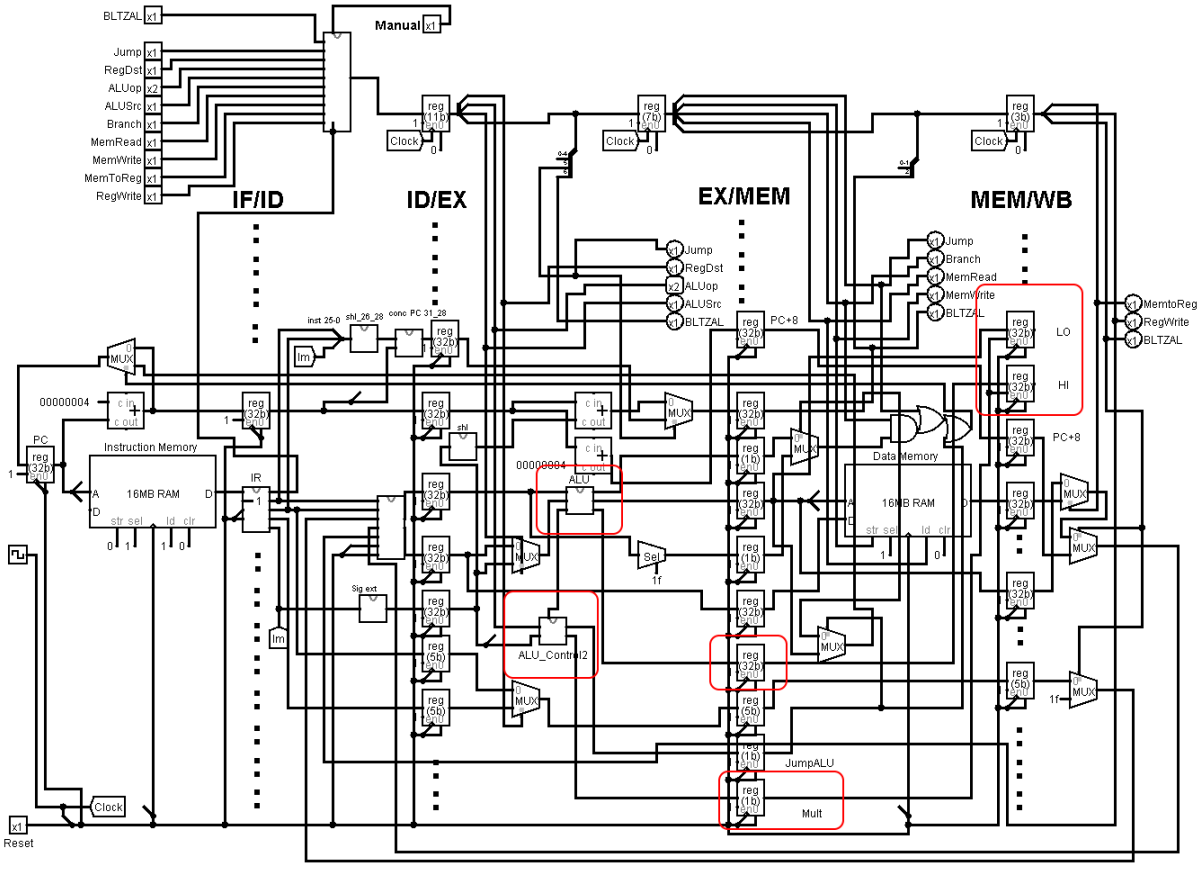
MULT

Modificações Estruturais

Foi introduzida uma nova lógica para o controle da ULA, assim como uma nova saída: Mult que em seguida é armazenada em um novo registrador.

Um multiplicador e uma nova saída foram introduzidos na ULA (MSB, 32 bits significantes da multiplicação).

Foram introduzidos dois registradores especiais LO e HI que dependem de ALUOut e MSB. Seu armazenamento é feito em MEM/WB e somente quando temos uma multiplicação (única operação que interage com LO e HI).



Modificações no Controle

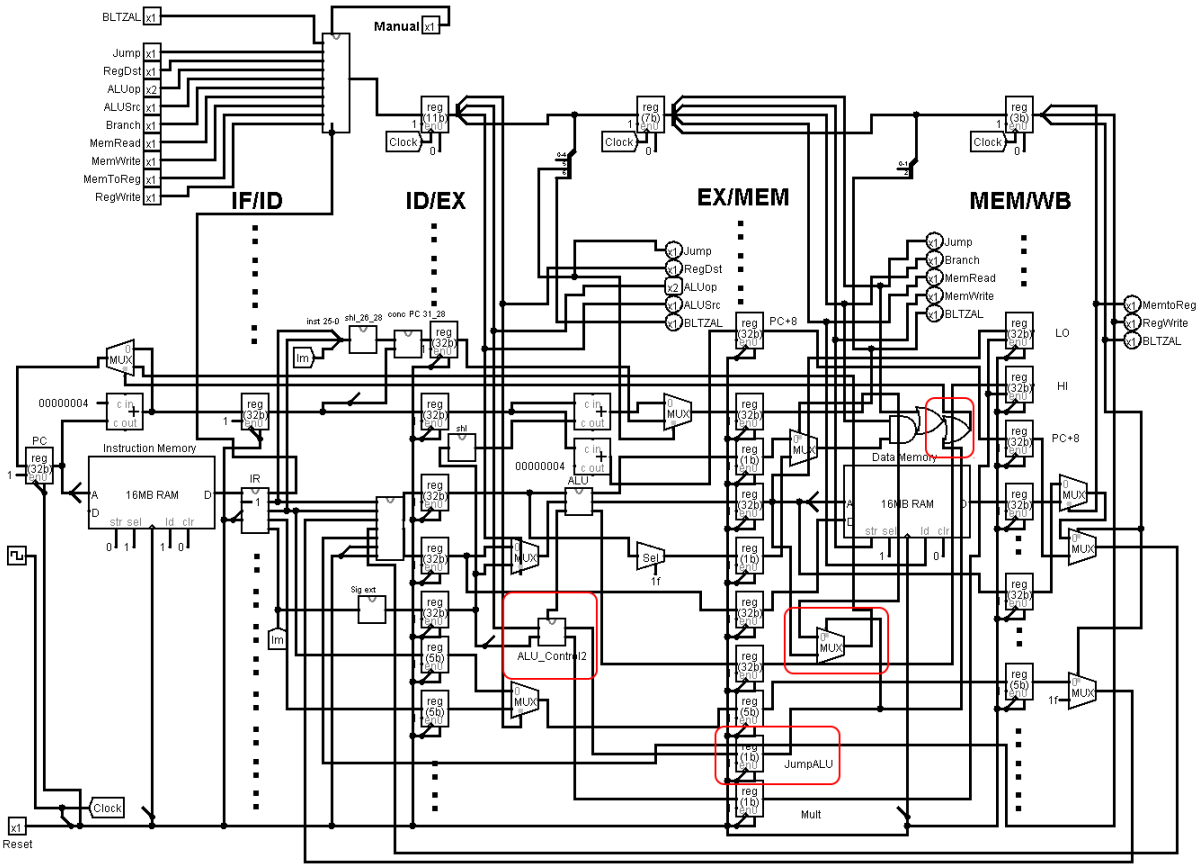
Em termos de controle, somente o controle da ULA foi alterado. Introduzimos uma saída relevante para instrução que somente é utilizada ao tivermos uma instrução do tipo MULT. Utilizamos um mapeamento de tabela de verdade para circuito vindo do próprio Logisim. A lógica é equivalente ao MULT do multiciclo usando operadores.

JR

Modificações Estruturais

Foi introduzida uma nova lógica para o controle da ULA, assim como uma nova saída: JumpALU que em seguida é armazenada em um novo registrador.

Foi introduzido um multiplexador controlado pelo bit derivado JumpALU para escolher se queremos pular para o resultado da ULA (já que a operação JR pode ser pensada como uma soma de rs+0). Também foi incluída uma porta OR com uma de suas entradas sendo JumpALU, isso garante que iremos fazer o pulo.



Modificações no Controle

Assim como o MULT, em termos de controle, somente o controle da ULA foi alterado. Introduzimos uma saída relevante para instrução que somente é utilizada ao tivermos uma instrução do tipo JR. Utilizamos um mapeamento de tabela de verdade para circuito vindo do próprio Logisim. A lógica é equivalente ao JR do multiciclo usando operadores.