Organização de Computadores B (2022/1)

Trabalho 1 - Implementação de Instruções MIPS

Grupo: Henrique Utzig, João P. Cosme, Matheus Stein de Aguiar e Victor Machado Gonçalves.

**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:

|  |
| --- |
| GPR[rt] ← GPR[rs] + || |

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

|  |  |
| --- | --- |
| GPR[ra] | Ler o registro ra do banco de registradores |
|  | Obter os bits de a (inclusive) da variável abc, podemos omitir y, levando somente a 1 bit. |
|  | Copiar sequência de bits da variável e unir sequencialmente vezes |
|  | Unir sequência de bits e , 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
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
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 ← || ||  condition ← ()  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 ←  LO ←  HI ← |

|  |
| --- |
| **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 ←  PC ← |

**MIPS – Monociclo**

***ADDI***

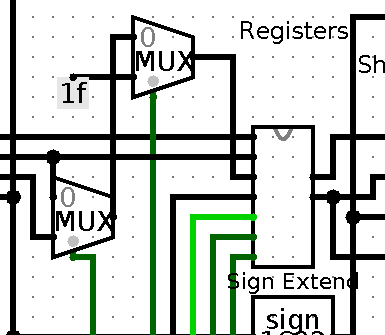
Não é necessário realizar nenhuma modificação estrutural, apenas no bloco de controle. A instrução ADDI é equivalente ao código 003 do bloco de comando, onde as únicas flags ativas são o AluSrc(precisa controlar a ULA pelo OpCode e não pelo funct visto que não é uma instrução R) e o RegWrite (precisa 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**

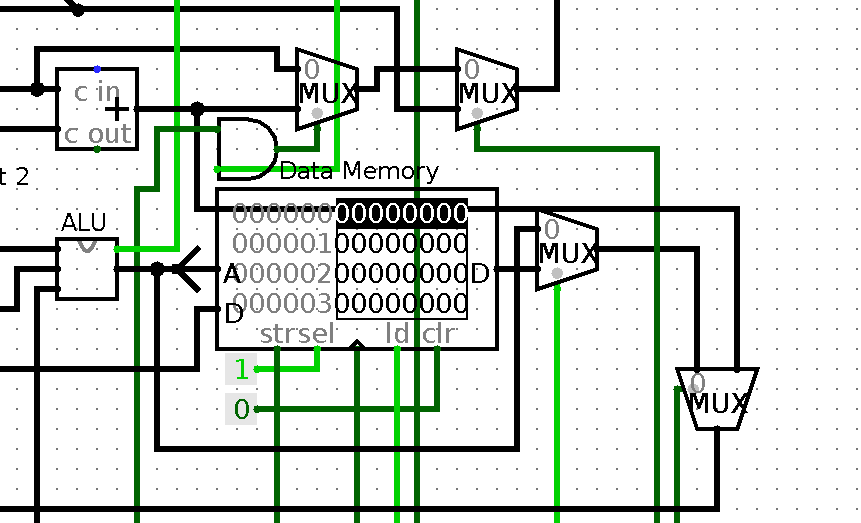
3 ações devem ser tomadas para a implementação do BLTZAL no Monociclo:

* Writeback no registrador 31 do valor de PC+4+Offset
* 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

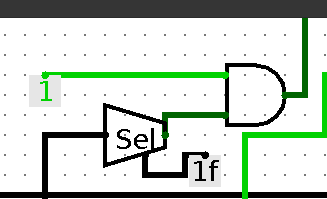
Para a primeira ação, utilizamos uma constante 31 e um novo mux na porta do banco de registradores. Este MUX não impacta diretamente no caminho critico já que, por estar no final do caminho, tem todo o ciclo para se estabilizar e não deve acarretar em um tempo maior de período necessário. Este MUX por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



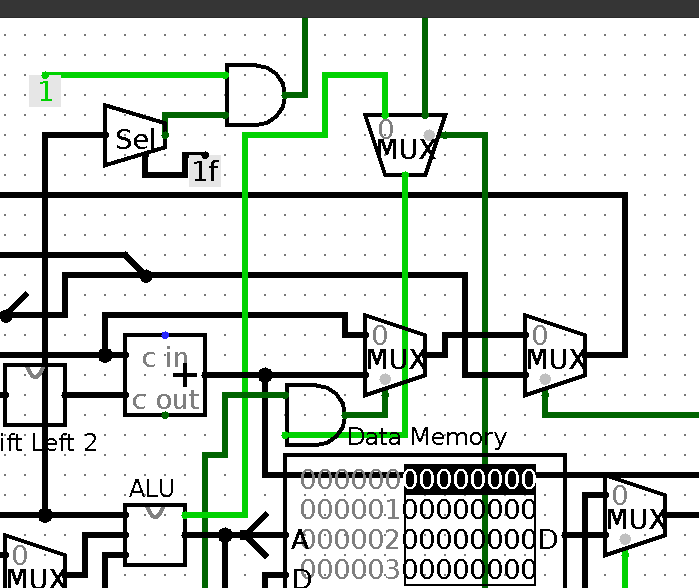
Para o envio do endereço de PC+4+Offset, ainda devemos enviar o resultado deste cálculo para o banco de registradores. Para isso, um novo MUX após a saída do banco de dados foi adicionado. Este MUX também é controlado por “Link” e, já que se incorpora ao caminho de LW, pode impactar negativamente o periodo 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.



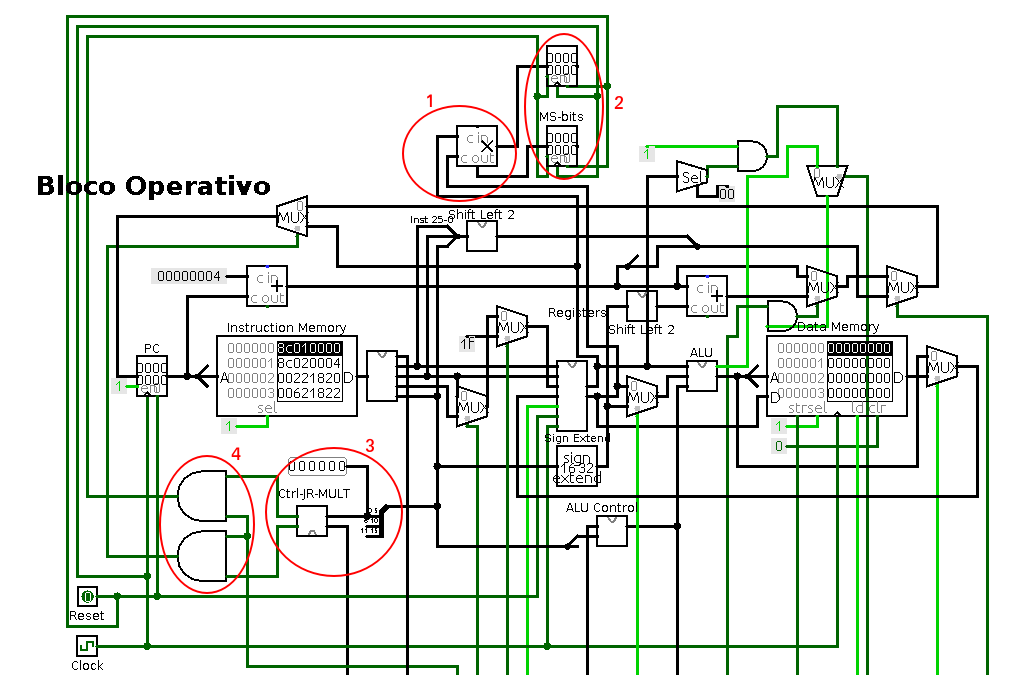
Por fim, um novo MUX é adicionado entre a saída “Zero” da ULA e a saida do novo comparador, isto nos garante que o bit enviado para ser comparado com o bit “branch” é o correto e não interferimos no funcionamento do BEQ no caso do valor do rs ser menor que zero. O tempo adicional deste MUX pode ser considerado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



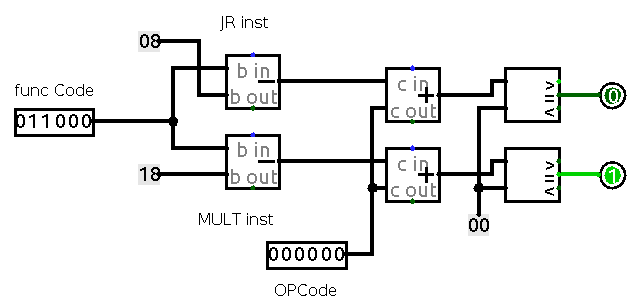
Os bits de controle, por fim, são idênticos ao de BEQ, com a diferença de que o bit de controle “Link” deve ser 1.

**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 ou 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;



1. 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.*

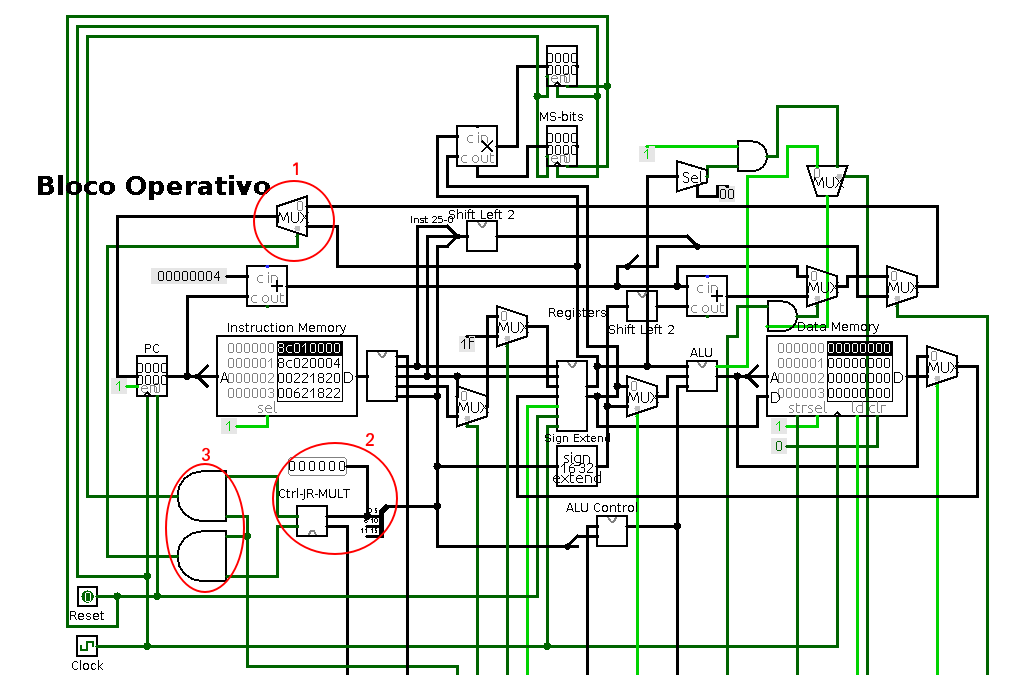
A instrução MULT funcionará com os seguintes sinais de controle:

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

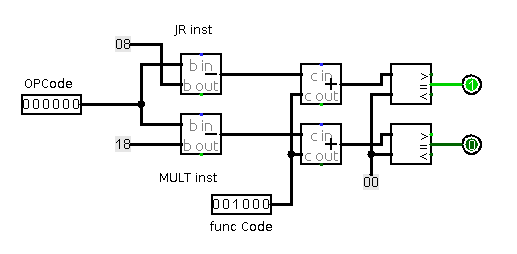
MULT não escrever em nenhum registrador presente no banco de registradores e nem na memória de dados então os sinais de escrita serão zerados (*RegWrite e MemWrite*). A instrução não realiza desvio, então os sinais *Jump e Branch* serão zerados, além disso, não utilizados a *ALU* para realizar a multiplicação 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;



1. 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.

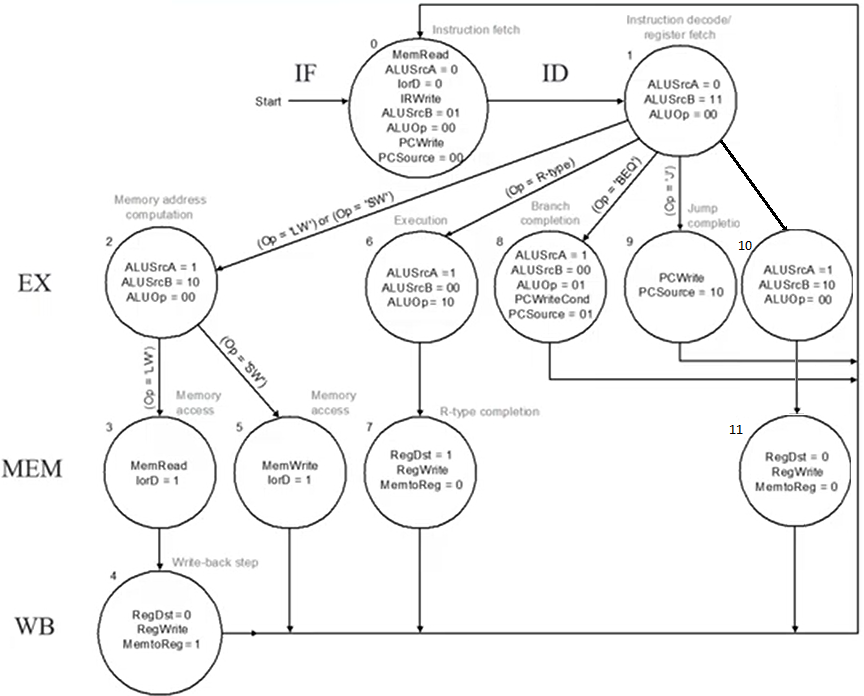
A instrução JR necessita dos seguintes sinais de controle para funcionar:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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** - Não é necessário adicionar nenhuma nova estrutura, mas é necessário adicionar novos estados a partir do estado 1, como pode ser visto na figura à seguir.



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** -

Alteraçao no estado 8:

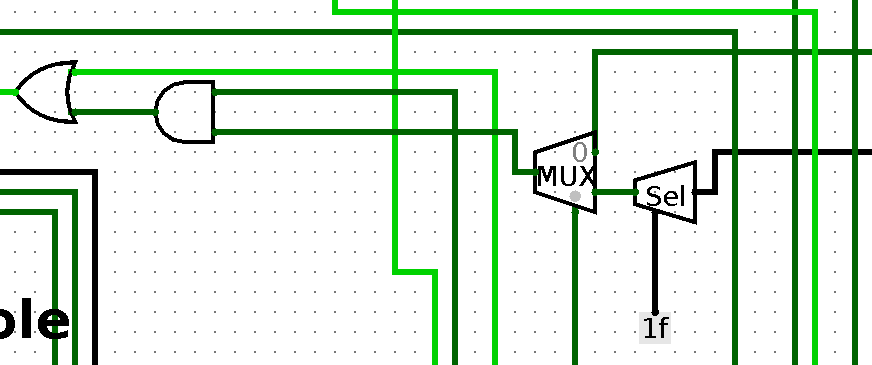
Link = 0

Novo estado 12:

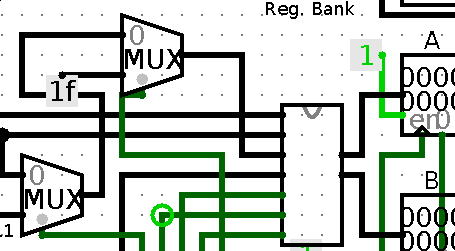
Link = 1

Comparador após o registrador A do banco de registradores para verificar se bit 31 == 1.

Novo mux para escolher entre zero e a saida do comparador de rs[31]



Enviar PC+4+offset para banco de registradores. Só é necessario escolher o endereço, o valor já estará em ALUOUT.



**MULT** -

**JR** -

**MIPS – Pipeline**

**ADDI** - Só é necessário configurar o bloco de 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).

**BLTZAL** -

**MULT** -

**JR** -