UNIVERSIDADE DO VALE DO ITAJAÍ (UNIVALI)
Centro de Ciências Tecnológicas da Terra e do Mar
Ciência da Computação - Arquitetura e Organização de Computadores
Prof. Douglas Rossi de Melo

Organização do MIPS monociclo no Quartus II

ANDRÉ LUIZ DA SILVA
IVAN CARLOS DOS SANTOS
RAFAEL CALADO

Itajaí - SC
14/06/2016

TÍTULO

Organização do MIPS monociclo no Quartus II

INTRODUÇÃO

Considerando o modelo fornecido do MIPS monociclo, neste trabalho acrescentamos suporte em hardware para execução das instruções Jump, Jump and Link e Jump Register, modificando o caminho de dados e o controle, e inserindo novos componentes. Ao término do projeto, validamos seu funcionamento através de um programa em linguagem de máquina extraído do simulador MARS.

As modificações bem como os resultados dos testes serão apresentadas com imagens comentadas, a fim de ilustrar e comprovar seu funcionamento.

TEMA

Inclusão de suporte em hardware às instruções Jump, Jump and Link e Jump Register no projeto do MIPS monociclo no Quartus II.

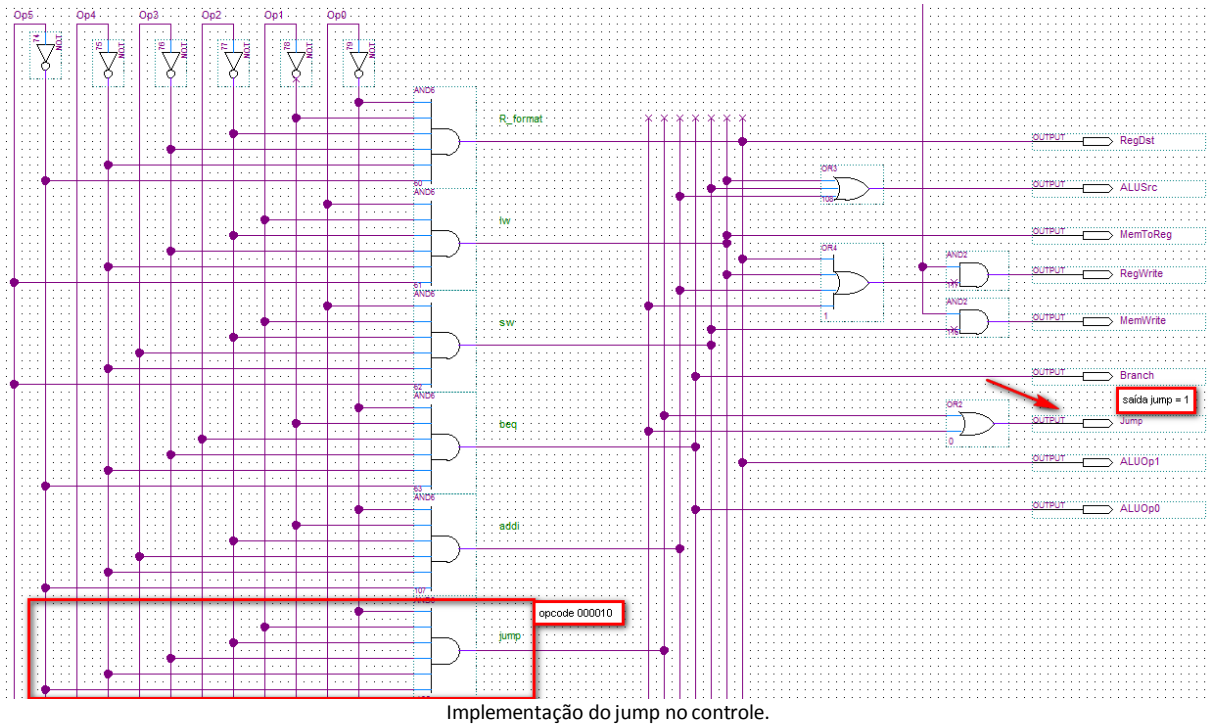
IMPLEMENTAÇÃO DA INSTRUÇÃO J

Para inclusão do suporte à instrução Jump, foi necessário incluir no *Control* uma verificação do OpCode da instrução com uma porta lógica AND, a fim de fornecer uma saída (output) 1 (verdadeiro) quando o OpCode for igual ao valor binário da instrução Jump (000010). A saída *jump* está conectada ao seletor de um *Mux 2x1 32 bits*, sendo que caso *jump* seja verdadeiro, a saída do Mux será o valor imediato da instrução Jump, que será carregado no PC a fim de executar em seguida a instrução solicitada.

Como o formato J possui 6 bits de OpCode + 26 bits de imediato, e o PC possui 32 bits, tivemos que deslocar os 26 bits do imediato duas vezes para a esquerda (acrescentando 2 bits “zero” à direita) e inserir à esquerda os 4 bits mais significativos do PC. Isto é necessário pois o assembler remove os últimos 2 bits (que serão sempre “00”) e os primeiros 4 bits (que serão sempre “0000” conforme formato do PC) do endereço para representa-lo em apenas 26 bits.

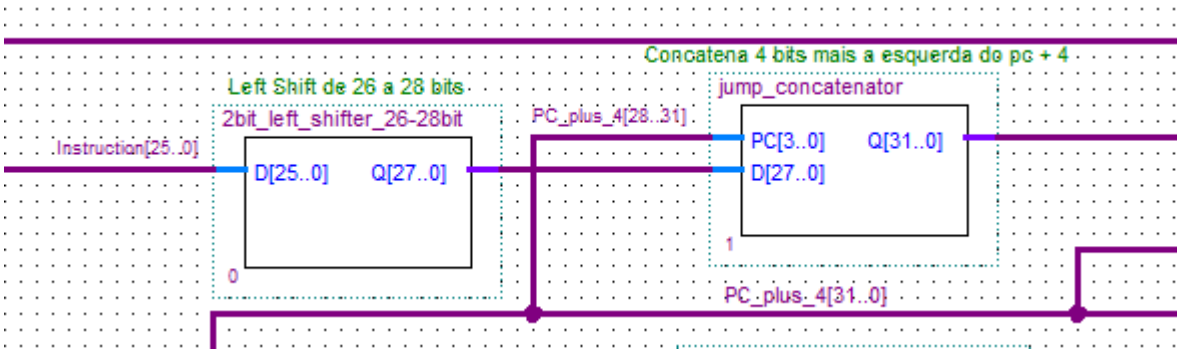
A seguir, ilustraremos com imagens a implementação da função Jump do MIPS monociclo no Quartus II:

JUMP #1 – Implementação da instrução Jump no controle:



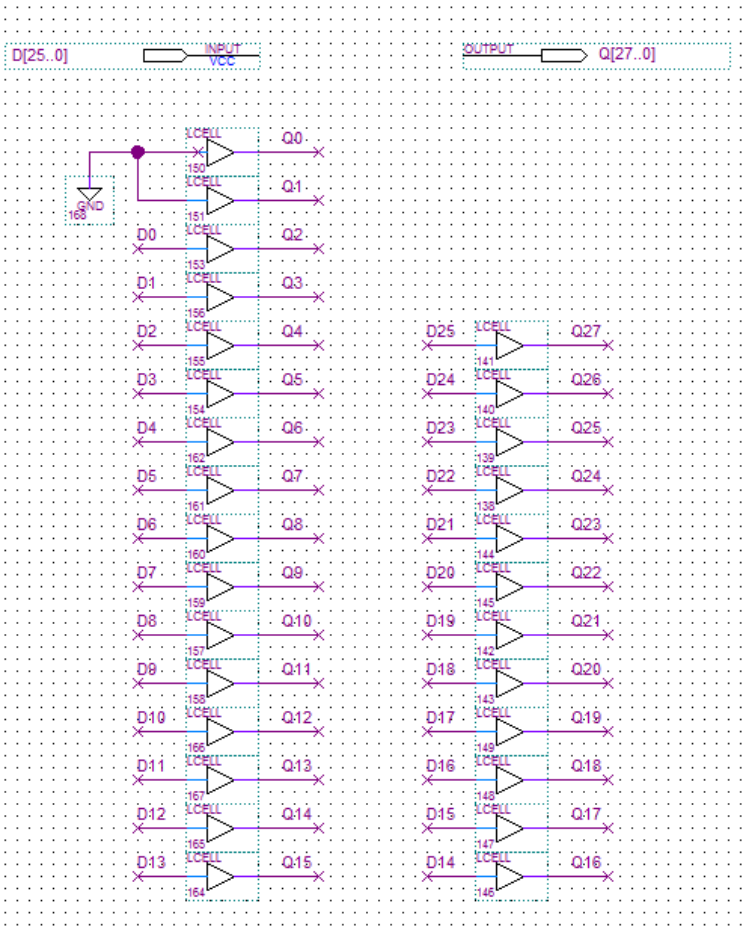
Implementação do jump no controle.

JUMP #2 – desloca imediato 2x para esquerda, e concatena com 4 bits do PC+4:



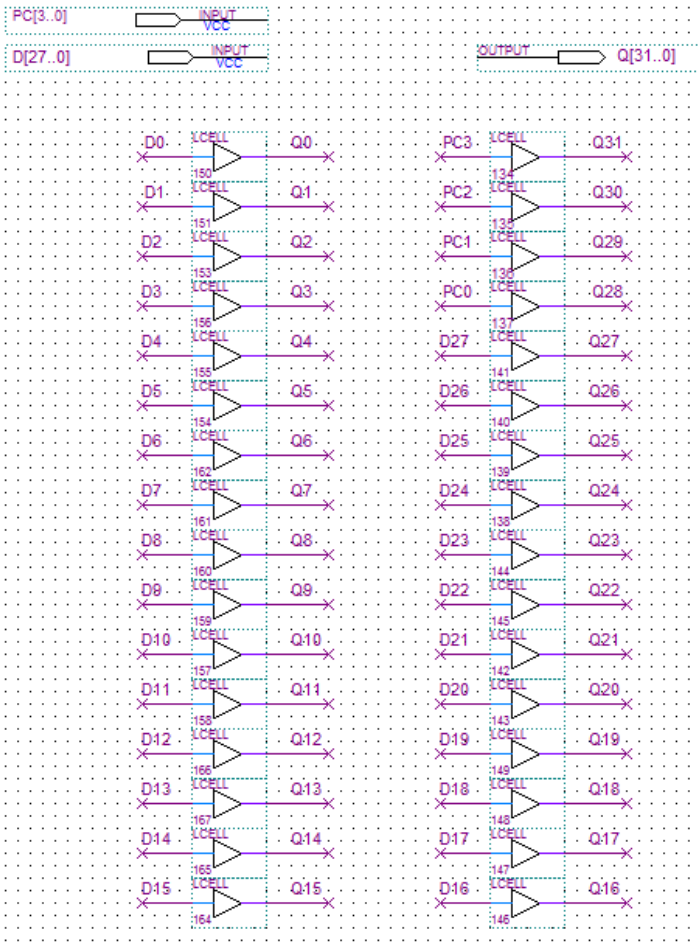
Insere 2 bits “zero” à direita tornando-o 28bits depois copia os 4 bits mais significativos do PC para a esquerda.

JUMP #2.1 – implementação do Left Shift de 26 a 28 bits:



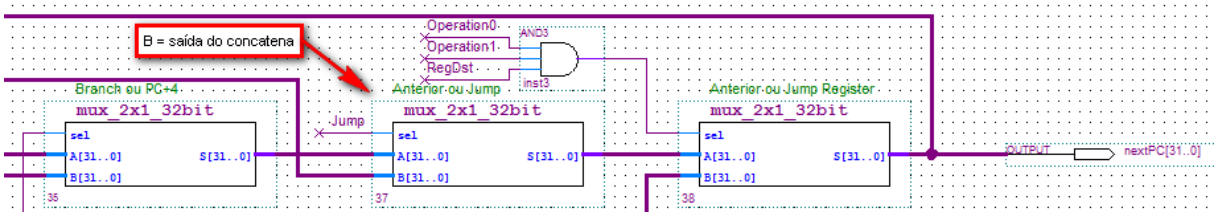
Acrescentado 2 bits “zero” (ground) à direita tornando a saída de 28 bits.

JUMP #2.2 – implementação do concatenador imediato + PC:



Acrescenta os 4 bits do PC+4 à esquerda e joga para a saída.

JUMP #3 – MUX do jump:



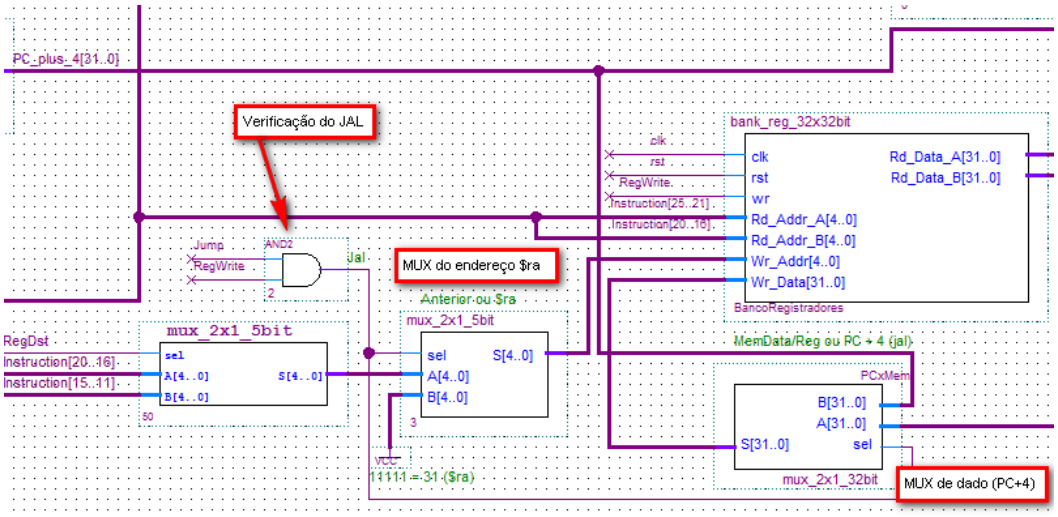
Como o seletor Jump é igual a 1, a saída do MUX é o endereço informado no imediato da instrução jump.

IMPLEMENTAÇÃO DA INSTRUÇÃO JAL

Para inclusão do suporte de hardware à instrução JAL, foi necessário – da mesma forma que na instrução jump – incluir uma nova verificação no controle utilizando uma porta lógica AND, para detectar o OpCode relativo à instrução Jump and Link (000011), e desta forma ativar as saídas (output) *jump* e *regWrite* com valor 1 (verdadeiro). A saída *jump* está conectada ao seletor de um *Mux 2x1 32 bits*, sendo que caso *jump* seja verdadeiro, a saída do Mux será o valor imediato da instrução Jump, que será carregado no PC a fim de executar em seguida a instrução solicitada. Porém, além de realizar o procedimento da função Jump, o Jump and Link grava no registrador $ra o valor do PC+4, portanto utilizamos um *Mux 2x1 5 bits* que utiliza como seletor a saída de uma porta AND com *jump* e *regWrite* – caso o valor de ambos sejam 1 (verdadeiro), a saída do Mux será o endereço do registrador $ra, registrador este que deverá receber o valor de PC+4, então outro Mux utilizando o mesmo seletor grava no banco de registradores, no endereço do registrador $ra, o valor de PC+4.

A seguir, ilustraremos com imagens as inclusões em relação ao Jump, para implementação da função Jump and Link do MIPS monociclo no Quartus II (utiliza também a implementação do Jump):

JAL #1 – Mux do Jump:



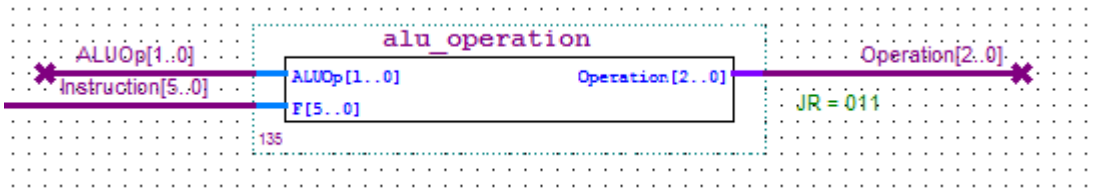
Grava o PC+4 no banco de registradores, no registrador $ra.

IMPLEMENTAÇÃO DA INSTRUÇÃO JUMP REGISTER

Para implementação do suporte em hardware para a instrução JR – diferentemente das instruções Jump e JAL – não foi necessário modificar o controle. Isso porque a instrução JR é uma instrução do formato R, e, portanto, sua "ativação" ocorre através do Function – os seis bits menos significativos da instrução. Verificamos a ocorrência do JR quando a saída do operation = 011 e o *RegDst* = 1 (verdadeiro) através de uma porta AND servindo como seletor de um *Mux 2x1 32bits*. Caso o JR seja "ativado", o Mux enviará para a saída o endereço recebido do registrador $ra, que então será carregado no PC.

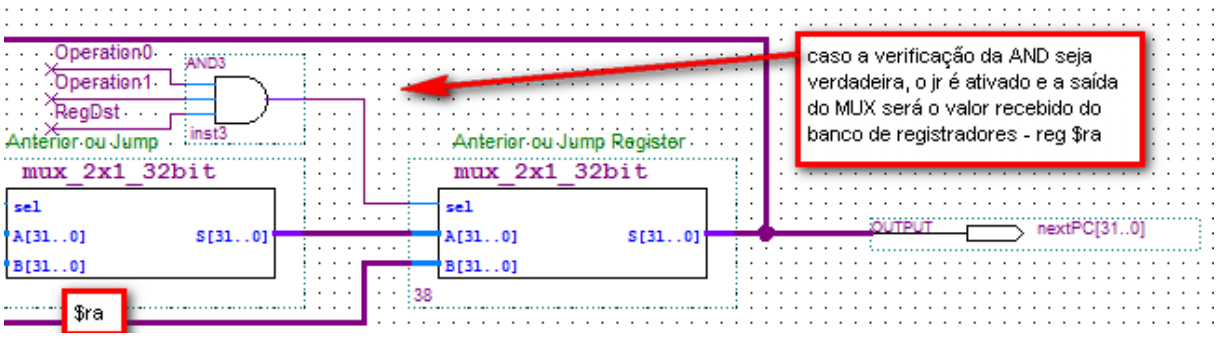
A seguir, ilustraremos com imagens as modificações para implementação da função Jump Register do MIPS monociclo no Quartus II:

JR #1 – ALU Operation:



Após o cálculo do alu_operation, a saída será 011 para função JR.

JR #2 – Mux do JR:



Saída do MUX do JR é o valor armazenado no registrador $ra, que será carregado no PC.

TESTE da implementação das funções j, jal e jr

Para validação da implementação das instruções j, jal e jr, utilizamos o programa fornecido leaf_example. O programa foi inserido no simulador MARS, e então foi gerado o DUMP para hexadecimal e incluído no rom.mif para execução no Quartus II. O resultado da simulação pode ser conferido abaixo.

Programa leaf_example.asm - assembly

```
# Trecho em C para validação do jr e do jal:
# int leaf_example (int g, int h, int i, int j) {
#    int f;
#    f = (g + h) - (i + j);
#    return f;

      .text   # segmento de código (programa)
       j   main
leaf_example:
       add $t0, $a0, $a1   # $t0 = g + h
       add $t1, $a2, $a3   # $t1 = i + j
       sub $v0, $t0, $t1   # f = $t0 - $t1
       jr  $ra             # retorna do procedimento

main:
       addi $a0, $zero, 4  # inicializa 1° parâmetro (g)
       addi $a1, $zero, 3  # inicializa 2° parâmetro (h)
       addi $a2, $zero, 2  # inicializa 3° parâmetro (i)
       addi $a3, $zero, 1  # inicializa 4° parâmetro (j)

       jal leaf_example    # chama o procedimento

       nop             # não faz nada. $v0 tem o resultado do procedimento
```

Programa leaf_example.mif – instruções em hexadecimal

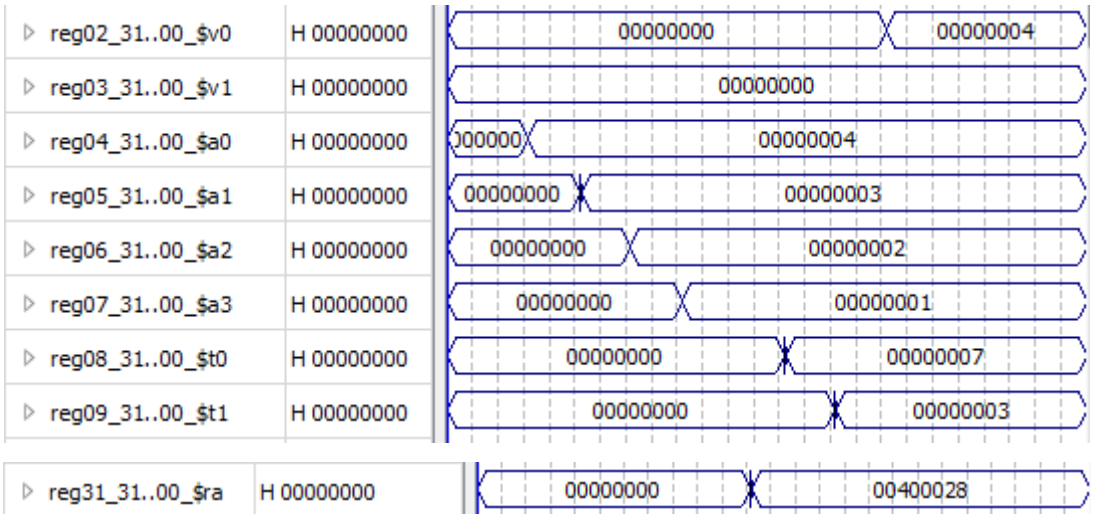
```
DEPTH = 256;    % Number of positions   %
WIDTH = 32; % Position size     %

ADDRESS_RADIX = HEX;
DATA_RADIX    = HEX;

CONTENT
  BEGIN
% The following addresses considers an offset that equals 0x04000020. In other
  words, the first position (0x00000000) points to 0x04000020, an so on. %
    00000000 : 08100005; % j main %
    00000001 : 00854020; % add $t0, $a0, $a1 %
    00000002 : 00c74820; % add $t1, $a2, $a3 %
    00000003 : 01091022; % sub $v0, $t0, $t1 %
    00000004 : 03e00008; % jr $ra %
    00000005 : 20040004; % addi $a0, $zero, 4 %
    00000006 : 20050003; % addi $a1, $zero, 3 %
    00000007 : 20060002; % addi $a2, $zero, 2 %
    00000008 : 20070001; % addi $a3, $zero, 1 %
    00000009 : 0c100001; % jal leaf_example %
    0000000A : 00000000; % nop %
    [0000000B..0000000F] : 00000000;
END ;
```

Resultado do programa – conteúdo dos registradores

Verificamos nas imagens abaixo, que os valores dos registradores utilizados no programa são iguais tanto na execução do simulador MARS quanto na execução da simulação no Quartus II, utilizando nossa implementação das funções Jump, Jump and Link e Jump Register.

Abaixo, o conteúdo dos registradores $v0 (retorno/resultado da operação $t0 - $t1), $a0, $a1, $a2, $a3, $t0 ($a0 + $a1) e $t1 ($a2 + $a3).

Quartus II:



MARS:

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000004
$v1	3	0x00000000
$a0	4	0x00000004
$a1	5	0x00000003
$a2	6	0x00000002
$a3	7	0x00000001
$t0	8	0x00000007
$t1	9	0x00000003
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00400028
pc		0x0040002c
hi		0x00000000
lo		0x00000000