UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

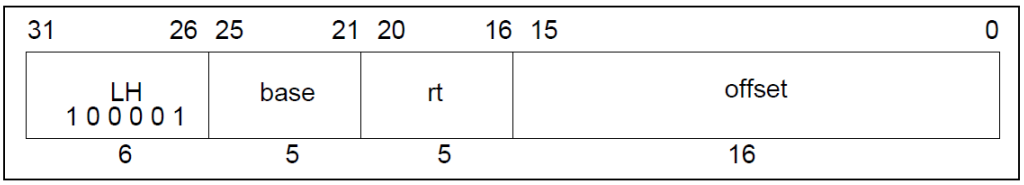
Departamento de Informática Aplicada

ORG B – INF01113

Prof. Dr. Antonio Carlos S. Beck Filho

Grupo 1 – Gabriel A. Zillmer, Gabriel Tiburski Júnior, Jean Andrade e Rodrigo Dal Ri

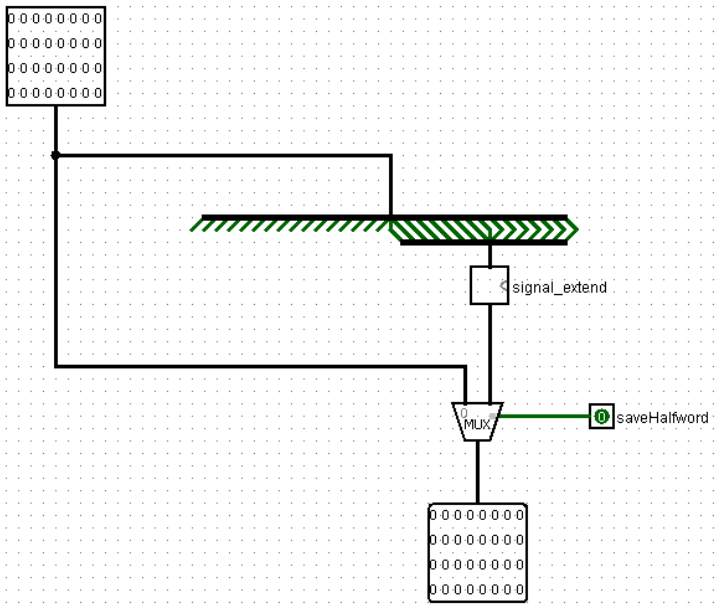
**LH** **Load Halfword** **LH**




Para implementação da função LH (Load Halfword) foi possível reaproveitar a função LW já presente nos circuitos disponibilizados, apenas com a adição de um componente e modificação nos bits de controle e estados.

Como, no nosso caso, podemos apenas buscar palavras de 32 bits na memória, a função LH que implementamos pega os 16 bits menos significativos da palavra e faz a extensão do sinal para 32 bits.

O componente de extensão de sinal já estava presente nas três versões e pôde ser reutilizado em outro novo, que utiliza um multiplexador para escolher a palavra normal ou a palavra transforma de acordo com um bit de controle, chamado saveHalfword. O componente criado é tal qual a imagem abaixo:



**Monociclo**

Primeiramente, para implementação no monociclo, foi necessário criar um bit de controle saveHalfword. Para isso, foi necessário aumentar o tamanho dos dados da ROM de controle em um bit e atualizar os tamanhos dos fios, multiplexadores e distribuidores no caminho. Os bits de controle de LH para a ROM, em hexadecimal, são 718.

Após isto, foi necessário apenas incluir o componente na entrada Write Data do banco de registradores e puxar o bit de controle até lá.

**Multiciclo**

Aqui, além de criar um bit de controle e aumentar os dados em 1 bit, foi necessário criar um estado novo também. Para a instrução de LH pegamos a instrução LW como base, que utilizava-se dos estados 1,2,3 e 4 da memória de geração de saída e trocamos o 4 por f (último endereço), que terá um valor parecido com o estado 4, mas com o bit mais significativo (saveHalfword, que criamos) como 1. [4: 00480 e f: 10480, em hexadecimal, com palavras de tamanho de 17 bits].

Feito isso, o novo componente foi incluído logo após a saída do registrador MDR (Memory Data Register), recebendo o bit de controle recém-criado. *(Diagrama de estados disponível na última página)*

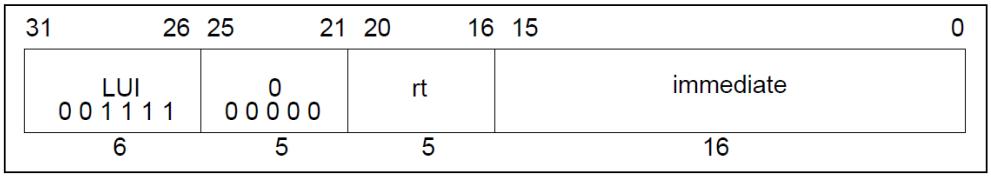
**Pipeline**

Novamente, começamos com o bit de controle. Após criá-lo e aumentar o tamanho dos dados da ROM, foi necessário alterar os dados para o opcode de LH [bin: 100001, hex: 21] para que tenha o valor dos de LW, mas com o novo bit em 1 [LW: 350, LH: 750].

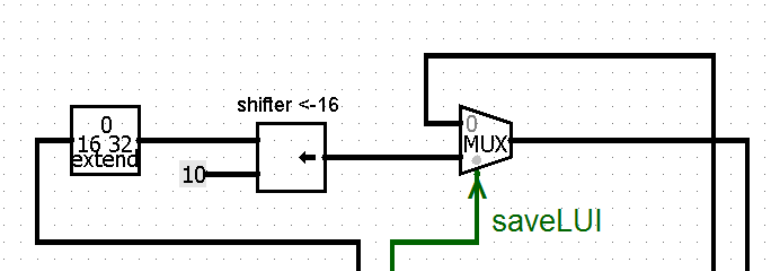
Após isso foi necessário passar o bit de controle por cada estágio do pipeline, pois ele será utilizado apenas no último, na hora de gravar os dados no banco de registradores.

Finalmente, como no monociclo, o novo componente é incluído na entrada Write Data do banco de registrados, ligado ao bit correspondente a saveHalfword do registrador do último estágio.

**LUI** **Load Upper Immediate** **LUI**




Para implementar esta instrução será necessário adicionar um hardware extra, mostrado abaixo.




Primeiro concatenamos o valor imediato com zeros, depois “shiftamos” 16 vezes para a esquerda. Para conseguirmos selecionar este valo para ser gravado no RT adicionamos um mux com um bit de controle chamado saveLUI que vai ser 1 somente nesta instrução.

**Monociclo**

| RegDST | ALUSrc | MemtoReg | RegWrite | MemRead | MemWrite | Branch | Jump | AluOp | saveLUI |
|---|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **00** | **1** |
| Savehal fword | JAL | | | | | | | | |
| **0** | **0** | | | | | | | | |

*Bits de controle para a instrução LUI - Monociclo*

**Multiciclo**

No multiciclo os dois primeiros passos são iguais para todas as instruções. Esta instrução terá apenas um passo adicional, neste passo adicional será calculado o valor (concatenar e shiftar) e será gravado no banco de registradores. *(Diagrama de estados disponível na última página)*

| PCWriteCond | PCWrite | lorD | MemRead | MemWrite | MemtoReg | IRWrite | RegDst | jal |
|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| saveHalfword | RegWrite | AluSrcA | AluSrcB | ALUop | PCSource | saveLUI | RegDst | |
| 0 | **1** | **0** | **00** | **00** | **00** | **1** | **0** | |

*Bits de controle para a instrução LUI - Multiciclo*

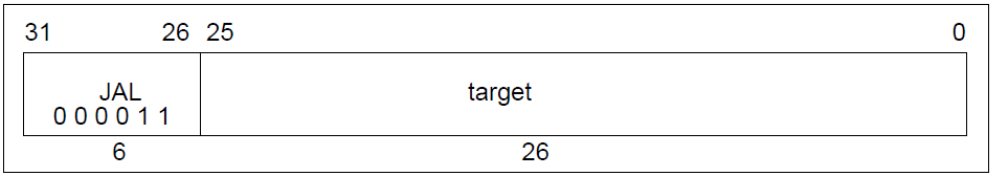
**Pipeline**

Foi necessário passar o bit de controle (saveLUI) por cada estágio do pipeline, pois ele será utilizado apenas no último, na hora de gravar os dados no banco de registradores.

| MemRead | MemWrite | MemtoReg | RegDst | RegWrite | AluSrc | ALUop | PCSource | saveLUI |
|---|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **1** | **0** | **00** | **0** | **1** |
| saveHalfword | JAL | | | | | | | |
| **0** | **0** | | | | | | | |

*Bits de controle para a instrução LUI - Pipeline*

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




Para implementação da função JAL (Jump and Link) foi possível aproveitar o jump já presente nos circuitos disponibilizados. Foi necessário apenas modificar os bits de controle e adicionar a opção de escolher o registrador 31 (RA - Return Address register) como write address.

**Monociclo**

Para implementar a instrução JAL no monociclo o foi necessário apenas adicionar um novo mux entre a saída do mux do RegDst e um fio constante com o endereço do registrador 31. O novo mux foi conectado no Write Data do banco de registradores e é controlado pelo bit JAL.

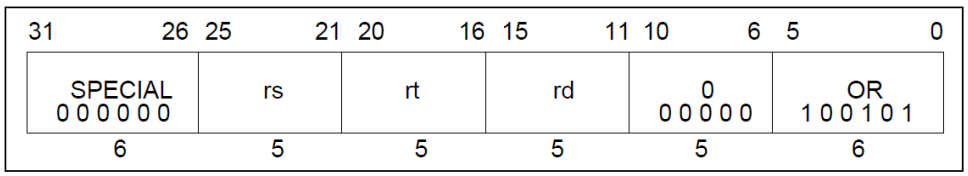
**Multiciclo**

No multiciclo, usamos os mesmos 2 passos iguais pra todas as instruções e adicionamos um novo passo. De modo que no primeiro a instrução é buscada, no segundo decodificada, no terceiro o registrador 31 recebe PC + 4 e o jump é realizado. *(Diagrama de estados disponível na última página)*

**Pipeline**

No pipeline foram adicionados 4 mux e alguns registradores. Os mux todos são controlados por um bit JAL e servem tanto para direcionar a escrita para o registrador 31, quanto para fazer o PC receber o endereço do jump. Os registradores foram usados para carregar o PC+4 até o estágio onde ele é necessário.

**OR** **Or** **OR**




A instrução OR é uma instrução que ocorre dentro da ULA do MIPS, e por isso, já vem implementada dentro do mesmo, não sendo necessária nenhuma alteração no hardware do sistema. Os bits de controle são gerados automaticamente nas três versões do MIPS, também não necessitando nenhuma alteração.

# Diagrama de Estados MIPS Multiciclo