UNIVERSIDADE FEDERAL DE PERNAMBUCO
CIn - CENTRO DE INFORMÁTICA

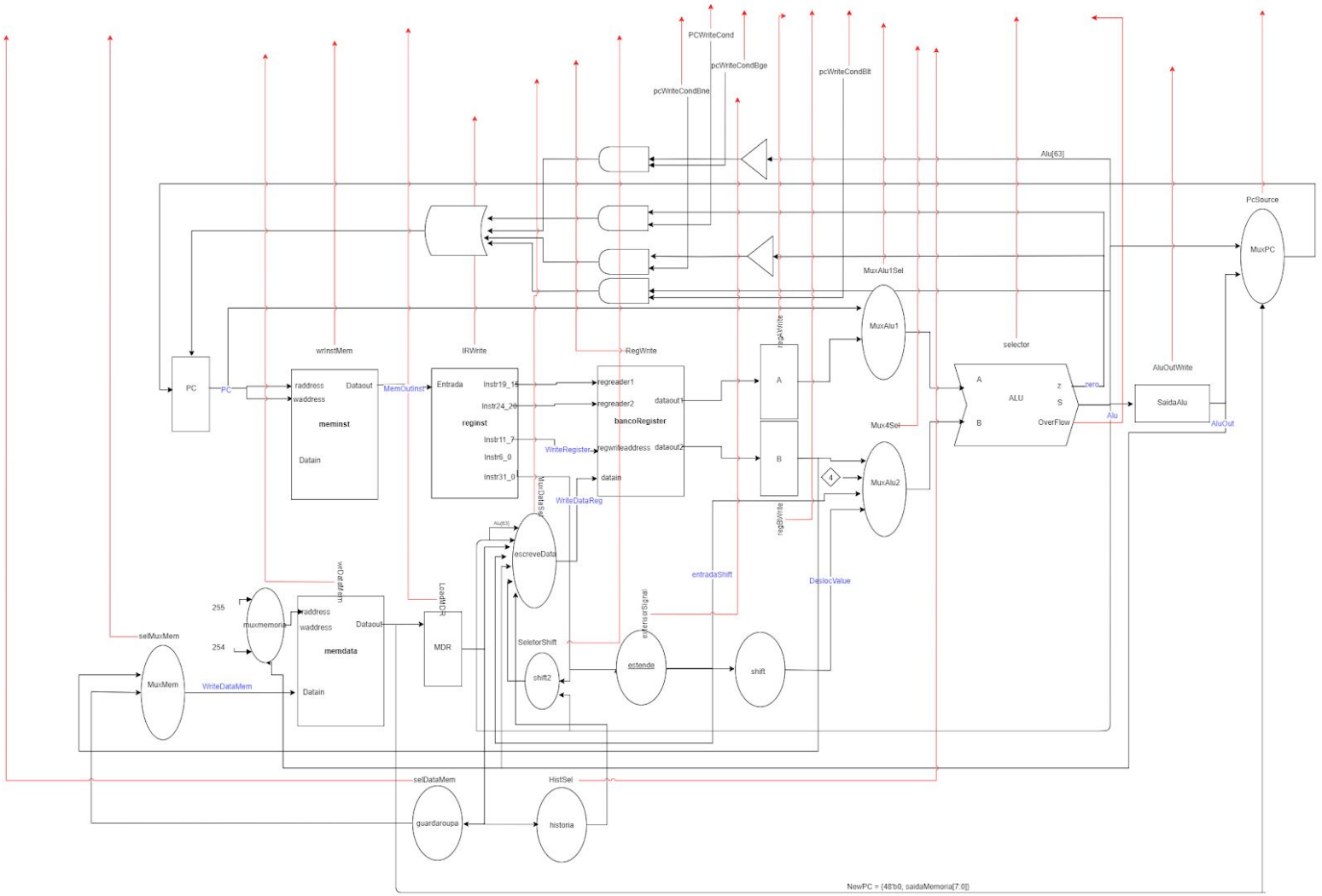
RELATÓRIO DO PROJETO

ELISSON RODRIGO DA SILVA ARAÚJO, ERSA
GABRIEL DE MELO EVANGELISTA, GME
JOÃO PEDRO SOUZA PEREIRA DE MOURA, JPSPM

RECIFE, 11 DE OUTUBRO DE 2019
PROFESSORA: EDNA NATIVIDADE DA SILVA BARROS

ÍNDICE

UNIDADE DE CONTROLE



DESCRIÇÃO DOS MÓDULOS

HISTORIADOR:

Entradas: [63:0]entrada, [4:0]sel.
Saídas: [63:0]saida.

O módulo historiador é utilizado nas instruções **Tipo I** (lb, lh, lw, lbu, lhu, lbu). O módulo tem como objetivo ser:

1. **Extensor de sinal:** Se a instrução for LB, LH ou LW o módulo estende o sinal da entrada, o seletor determina qual instrução está sendo executada e isso diferencia a forma da extensão.
   Na instrução LB o módulo verifica o bit que está em entrada[7].
   - Se o bit for 1, a saída do módulo é concatenação de: *saida={56'b11111111111111111111111111111111111111111111111111111111, entrada[7:0]}*.
   - Se o bit for 0, a saida do modulo é a concatenação de: *saida={56'd0, entrada[7:0]}*.

   Na instrução LH o módulo verifica o bit que está em entrada[15].
   - Se o bit for 1, a saída do módulo é a concatenação de: *saida={48'b111111111111111111111111111111111111111111111111, entrada[15:0]}*
   - Se o bit for 0, a saída do módulo é a concatenação de: *saida={48'd0, entrada[15:0]}*.

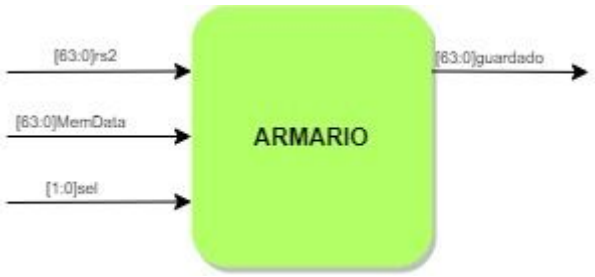
   Na instrução LW o módulo verifica o bit que está em entrada[31].
   - Se o bit for 1, a saída do módulo é a concatenação de: *saida={32 'b11111111111111111111111111111111, entrada[31:0]}*.
   - Se o bit for 0, a saída do módulo é a concatenação de: *saida={32'd0, entrada[31:0]}*.

2. **Complementar com zero:** Se a instrução for LBU, LHU ou LWU o módulo complementa a entrada com zero, pois se trata de instruções unsigned.
   Na instrução LBU a saída do módulo é a concatenação de:
   saida={56'd0, entrada[7:0]}
   Na instrução LHU a saída do módulo é a concatenação de:
   saida={48'd0, entrada[15:0]}.

Na instrução LWU a saída do módulo é a concatenação de:

saida={32'd0, entrada[31:0]}.

ARMARIO:

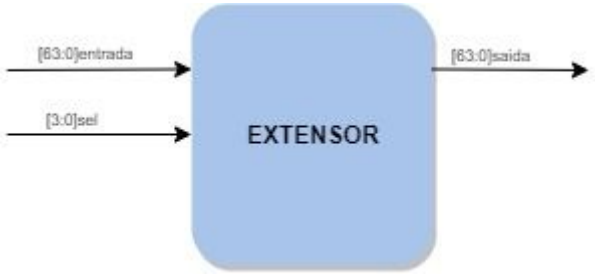


Entradas: [63:0]rs2, [63:0]MemData [4:0]sel.
Saídas: [63:0]saida.

O módulo armario é utilizado nas instruções **Tipo S** (sw, sh, sb). O módulo tem a seguinte função:

1. **Extensor de sinal:** Se a instrução for LB, LH ou LW o módulo estende o sinal da entrada, o seletor determina qual instrução está sendo executada e isso diferencia a forma da extensão.

EXTENSOR:



Entradas: [63:0]entrada, [3:0]sel.
Saídas: [63:0]saida.

O módulo extensor é utilizado para modularizar as instruções que precisam de algum tipo de extensão. O extensor é usado pela seguintes instruções:

1. **ADDI:** Recebe o imediato de 32 bits e faz a extensão do sinal do imediato para 64bits e verifica o bit entrada[31]:
   - Se o bit for 0, a saída do extensor é:
     saida = {52'd0, entrada[31:20]};
   - Se o bit for 1, a saída do extensor é:
     saida = {52'hFFFFFFFFFFFFF, entrada[31:20]};
2. **LOAD:** Recebe o imediato de 32 bits e faz a extensão do sinal do imediato para 64bits e verifica o bit entrada[31]:
   - Se o bit for 0, a saída do extensor é:
     saida = {52'd0, entrada[31:20]};
   - Se o bit for 1, a saída do extensor é:
     saida = {52'hFFFFFFFFFFFFF, entrada[31:20]};
3. **SD:** Recebe o imediato de 32 bits e faz a extensão do sinal do imediato para 64bits e verifica o bit entrada[31]:
   - Se o bit for 0, a saída do extensor é:
     saida = {52'd0,entrada[31:25], entrada[11:7]};
   - Se o bit for 1, a saída do extensor é:
     saida = {52'hFFFFFFFFFFFFF, entrada[31:25], entrada[11:7]};
4. **LUI:** Recebe o imediato de 32 bits e faz a extensão do sinal do imediato para 64bits e verifica o bit entrada[31]:
   - Se o bit for 0, a saída do extensor é:
     saida = {32'b0, entrada[31:12], 12'b0};
   - Se o bit for 1, a saída do extensor é:
     saida = {32'b11111111111111111111111111111111, entrada[31:12], 12'b0};
5. **SLTI:** Recebe o imediato de 32 bits e faz a extensão do sinal do imediato para 64bits e verifica o bit entrada[31]:
   - Se o bit for 0, a saída do extensor é:
     saida = {52'd0, entrada[31:20]};
   - Se o bit for 1, a saída do extensor é:
     saida = {52'hFFFFFFFFFFFFF, entrada[31:20]};

6. **JALR:** Recebe o imediato de 32 bits e faz a extensão do sinal do imediato para 64bits e verifica o bit entrada[31]:
    - Se o bit for 0, a saída do extensor é:
      saida = {52'd0, entrada[31:20]};
    - Se o bit for 1, a saída do extensor é:
      saida = {52'hFFFFFFFFFFFFF, entrada[31:20]};
7. **JAL:** Recebe o imediato de 32 bits e faz a extensão do sinal do imediato para 64bits e verifica o bit entrada[31]:
    - Se o bit for 0, a saída do extensor é:
      saida = {43'b0,entrada[31],entrada[19:12],entrada[20], entrada[30:21], 1'b0};
    - Se o bit for 1, a saída do extensor é:
      saida = {43'b11111111111111111111111111111111111111111, entrada[31], entrada[19:12], entrada[20], entrada[30:21], 1'b0};
8. **SW/SH/SB:** Recebe o imediato de 32 bits e faz a extensão do sinal do imediato para 64bits e verifica o bit entrada[31]:
    - Se o bit for 0, a saída do extensor é:
      saida = {52'd0,entrada[31:25], entrada[11:7]};
    - Se o bit for 1, a saída do extensor é:
      saida = {52'hFFFFFFFFFFFFF, entrada[31:25], entrada[11:7]};
9. **BEQ:** Recebe o imediato de 32 bits e faz a extensão do sinal do imediato para 64bits e verifica o bit entrada[31]:
    - Se o bit for 0, a saída do extensor é:
      saida = {52'd0, entrada[31], entrada[7], entrada[30:25], entrada[11:8], 1'b0};
    - Se o bit for 1, a saída do extensor é:
      saida = {52'hFFFFFFFFFFFFF, entrada[31], entrada[7], entrada[30:25], entrada[11:8], 1'b0};

DESCRIÇÃO DAS OPERAÇÕES

Instrução: *add rd, rs1, rs2*

Após identificar a instrução add no estágio de decodificação os registradores rs1 e rs2 são carregados e a operação de soma é feita na ula, esse valor é escrito no registrador AluOut e no próximo estado o registrador rd é escrito com o valor contido em AluOut e uma nova instrução é carregada

Instrução: *sub rd, rs1, rs2*

Após identificar a instrução sub no estágio de decodificação os registradores rs1 e rs2 são carregados e a operação de subtração é feita na ula, esse valor é escrito no registrador AluOut e no próximo estado o registrador rd é escrito com o valor contido em AluOut e uma nova instrução é carregada

Instrução: *addi rd, rs1, imm*

Após identificar a instrução addi no estágio de decodificação o registrador rs1 é carregado e o imediato é estendido para 64 bits e a operação de soma é feita na ula dos dois valores, o resultado fica salvo no registrador rd e uma nova instrução é carregada

Instrução: *ld rd, imm(rs1)*

Após identificar a instrução ld no estágio de decodificação o registrador rs1 é carregado, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a leitura da memória a partir do registrador AluOut, após isso o registrador MDR é carregado com o conteúdo lido da memória e por fim este valor é escrito no registrador rd e uma nova instrução é carregada

Instrução: *sd rs2, imm(rs1)*

Após identificar a instrução sd no estágio de decodificação os registradores rs1 e rs2 são carregados, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a escrita do registrador rs2 a partir do endereço de memória no registrador AluOut e uma nova instrução é carregada

Instrução: *bne rs1, rs2, imm*

Após identificar a instrução bne no estágio de decodificação os registradores rs1 e rs2 são carregados e o novo valor de PC é calculado e armazenado no registrador AluOut, no próximo estado é feita a comparação do registrador rs1 e do registrador rs2 e a flag pcWriteCondBne é setada para 1, caso rs1 seja diferente de rs2 o valor no registrador AluOut será escrito no registrador PC e uma nova instrução é carregada

Instrução: *beq rs1, rs2, imm*

Após identificar a instrução beq no estágio de decodificação os registradores rs1 e rs2 são carregados e o novo valor de PC é calculado e armazenado no registrador AluOut, no próximo estado é feita a comparação do registrador rs1 e do registrador rs2 e a flag pcWriteCond é setada para 1, caso rs1 seja igual rs2 o valor no registrador AluOut será escrito no registrador PC e uma nova instrução é carregada

Instrução: *lui rd, imm*

Após identificar a instrução lui no estágio de decodificação, o imediato é estendido e esse valor é escrito no registrador rd e uma nova instrução é carregada e uma nova instrução é carregada

Instrução: *and rd, rs1, rs2*

Após identificar a instrução and no estágio de decodificação os registradores rs1 e rs2 são carregados e a operação de and é feita na ula, esse valor é escrito no registrador AluOut e no próximo estado o registrador rd é escrito com o valor contido em AluOut e uma nova instrução é carregada

Instrução: *slt rd, rs1, rs2*

Após identificar a instrução slt no estágio de decodificação os registradores rs1 e rs2 são carregados, no próximo estado ocorre a operação de subtração do rs1 com o rs2 na ula e o bit 63 do resultado é escrito no registrador rd e uma nova instrução é carregada

Instrução: *stli rd, rs1, imm*

Após identificar a instrução slti no estágio de decodificação o registrador rs1 é carregado, no próximo estado ocorre a operação de subtração do rs1 com o imediato estendido para 64 bits na ula e o bit 63 do resultado é escrito no registrador rd e uma nova instrução é carregada

Instrução: *jalr rd, rs1, imm*

Após identificar a instrução slti no estágio de decodificação o registrador rs1 é carregado, no próximo estado ocorre a escrita do valor de PC no registrador rd, no próximo estado o novo valor de PC é calculado a partir da soma do registrador PC com o imediato estendido para 64 bits na ula e esse resultado é escrito no registrador AluOut, no próximo estado ocorre a escrita no registrador PC do valor em AluOut e uma nova instrução é carregada

Instrução: *lb rd, imm(rs1)*

Após identificar a instrução lb no estágio de decodificação o registrador rs1 é carregado, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a leitura da memória a partir do registrador AluOut, após isso o registrador MDR é carregado com o conteúdo lido da memória e por fim o valor de MDR passa pelo módulo "historia", que irá estender com sinal o byte[7:0], então o valor que sai deste módulo é escrito no registrador rd e uma nova instrução é carregada

Instrução: *lh rd, imm(rs1)*

Após identificar a instrução lh no estágio de decodificação o registrador rs1 é carregado, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a leitura da memória a partir do registrador AluOut, após isso o registrador MDR é carregado com o conteúdo lido da memória e por fim o valor de MDR passa pelo módulo "historia", que irá estender com sinal a halfword[15:0], então o valor que sai deste módulo é escrito no registrador rd e uma nova instrução é carregada.

Instrução: *lw rd, imm(rs1)*

Após identificar a instrução lw no estágio de decodificação o registrador rs1 é carregado, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a leitura da memória a partir do registrador AluOut, após isso o registrador MDR é carregado com o conteúdo lido da memória e por fim o valor de MDR passa pelo módulo "historia", que irá estender com sinal a word[31:0], então o valor que sai deste módulo é escrito no registrador rd e uma nova instrução é carregada.

Instrução: *lbu rd, imm(rs1)*

Após identificar a instrução lbu no estágio de decodificação o registrador rs1 é carregado, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a leitura da memória a partir do registrador AluOut, após isso o registrador MDR é carregado com o conteúdo lido da memória e por fim o valor de MDR passa pelo módulo "historia", que irá estender com 0 o byte[7:0], então o valor que sai deste módulo é escrito no registrador rd e uma nova instrução é carregada.

Instrução: *lhu rd, imm(rs1)*

Após identificar a instrução lhu no estágio de decodificação o registrador rs1 é carregado, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a leitura da memória a partir do registrador AluOut, após isso o registrador MDR é carregado com o conteúdo lido da memória e por fim o valor de MDR passa pelo módulo "historia", que irá estender com 0 a halfword[15:0], então o valor que sai deste módulo é escrito no registrador rd e uma nova instrução é carregada.

Instrução: *lwu rd, imm(rs1)*

Após identificar a instrução lwu no estágio de decodificação o registrador rs1 é carregado, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a leitura da memória a partir do registrador AluOut, após isso o registrador MDR é carregado com o conteúdo lido da memória e por fim o valor de MDR passa pelo módulo "historia", que irá estender com 0 a word[31:0], então o valor que sai deste módulo é escrito no registrador rd e uma nova instrução é carregada.

Instrução: *nop*

Após identificar a instrução nop no estágio de decodificação, voltamos para o estado de busca.

Instrução: *break*

Após identificar a instrução break no estágio de decodificação, o código é colocado num loop infinito.

Instrução: *srli rd, rs1, shamt*

Após identificar a instrução srli no estágio de decodificação o registrador rs1 é carregado, o registrador rs1 após dado o shift right lógico shamt vezes é escrito no registrador rd e uma nova instrução é carregada.

Instrução: *srai rd, rs1, shamt*

Após identificar a instrução srai no estágio de decodificação o registrador rs1 é carregado, o registrador rs1 após dado o shift right aritmético shamt vezes é escrito no registrador rd e uma nova instrução é carregada.

Instrução: *slli rd, rs1, shamt*

Após identificar a instrução slli no estágio de decodificação o registrador rs1 é carregado, o registrador rs1 após dado o shift left lógico shamt vezes é escrito no registrador rd e uma nova instrução é carregada.

Instrução: *sw rs2, imm(rs1)*

Após identificar a instrução sw no estágio de decodificação os registradores rs1 e rs2 são carregados, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a leitura da memória a partir do registrador AluOut, após isso o registrador MDR é carregado com o conteúdo lido da memória, esse conteúdo é então passado pelo módulo "armario", que irá concatenar o conteúdo da memória de [63:32] com o registrador rs2 de [31:0], a saída do "armario" é então escrita no endereço de memória de valor AluOut e uma nova instrução é carregada.

Instrução: *sh rs2, imm(rs1)*

Após identificar a instrução sh no estágio de decodificação os registradores rs1 e rs2 são carregados, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a leitura da memória a partir do registrador AluOut, após isso o registrador MDR é carregado com o conteúdo lido da memória, esse conteúdo é então passado pelo módulo "armario", que irá concatenar o conteúdo da memória de [63:16] com o registrador rs2 de [15:0], a saída do "armario" é então escrita no endereço de memória de valor AluOut e uma nova instrução é carregada.

Instrução: *sb rs2, imm(rs1)*

Após identificar a instrução sb no estágio de decodificação os registradores rs1 e rs2 são carregados, o endereço de memória é então obtido a partir da soma do registrador rs1 e o imediato estendido para 64 bits, esse resultado é escrito no registrador AluOut, no próximo estado é feito a leitura da memória a partir do registrador AluOut, após isso o registrador MDR é carregado com o conteúdo lido da memória, esse conteúdo é então passado pelo módulo "armario", que irá concatenar o conteúdo da memória de [63:8] com o registrador rs2 de [7:0], a saída do "armario" é então escrita no endereço de memória de valor AluOut e uma nova instrução é carregada.

Instrução: *bge rs1, rs2, imm*

Após identificar a instrução bge no estágio de decodificação os registradores rs1 e rs2 são carregados e o novo valor de PC é calculado e armazenado no registrador AluOut, no próximo estado é feita a comparação do registrador rs1 e do registrador rs2 e a flag pcWriteCondBge é setada para 1, caso rs1 seja maior igual a rs2 o valor no registrador AluOut será escrito no registrador PC e uma nova instrução é carregada.

Instrução: *blt rs1, rs2, imm*

Após identificar a instrução blt no estágio de decodificação os registradores rs1 e rs2 são carregados e o novo valor de PC é calculado e armazenado no registrador AluOut, no próximo estado é feita a comparação do registrador rs1 e do registrador rs2 e a flag pcWriteCondBlt é setada para 1, caso rs1 seja menor que rs2 o valor no registrador AluOut será escrito no registrador PC e uma nova instrução é carregada.

Instrução: *jal rd, imm*

Após identificar a instrução jal no estágio de decodificação, no próximo estado de execução o valor de PC é escrito no registrador rd, no próximo estado calculamos o novo valor de PC pela soma de PC com o imediato estendido para 64 bits esse resultado é escrito no registrador AluOut, no próximo estado ocorre a escrita de PC com o valor armazenado no registrador AluOut e uma nova instrução é carregada.

DESCRIÇÕES DOS ESTADOS DE CONTROLE

Estado 0:

No estado 0 ocorre a colocação de todos os seletores para 0, como Muxes e os bits de escrita/leitura de componentes do CPU, etc.

Estado 1:

No estado 1 faz-se a requisição da leitura na memória do endereço da instrução armazenado no registrador PC e atualiza-se o mesmo com o que tem nele próprio mais quatro.

Estado 2:

Neste estado ocorre o carregamento dos registradores A e B para estarem disponíveis no próximo estado e o valor de PC + imm é carregado na Ula e o seu valor é guardado no registrador AluOut. Além disso, ocorre a decodificação da instrução para determinar qual será o próximo estado.

Estado 3(addi):

Neste estado ocorre a soma do valor no registrador A com o imm estendido e o valor é guardado no registrador AluOut e estará disponível no próximo estado. Além disso, temos a verificação de OverFlow.

Estado 4(add):

Neste estado ocorre a soma do valor no registrador A com o valor no registrador B e o valor é guardado no registrador AluOut e estará disponível no próximo estado. Além disso, temos a verificação de OverFlow.

Estado 5(sub):

Neste estado ocorre a subtração do valor no registrador A com o valor no registrador B e o valor é guardado no registrador AluOut e estará disponível no próximo estado. Além disso, temos a verificação de OverFlow.

Estado 6(load):

Neste estado ocorre o cálculo do endereço, a partir da soma do valor no registrador A com o Imm estendido e o valor é guardado no registrador AluOut e estará disponível no próximo estado. Além disso, temos a verificação de OverFlow.

Estado 7:

Neste estado ocorre a comparação de 3 bits, localizados na instrução e decide qual operação fazer. lb, lh, lw, lbu, lhu, lwu, ld e ocorre o carregamento do registrador MDR.

Estado 8:

Neste estado carregamos no registrador destino o valor armazenado no registrador MDR.

Estado 9:

Neste estado ocorre o cálculo do endereço, a partir da soma do valor no registrador A com o Imm estendido e o valor é guardado no registrador AluOut e estará disponível no próximo estado. Além disso, temos a verificação de OverFlow.

Estado 10:

Neste estado solicitamos a leitura da memória a partir do endereço armazenado em AluOut, estará disponível no próximo estado.

Estado 11(beq):

Neste estado realizamos a comparação de igual do valor no registrador A com o valor no registrador B e setamos os valores de PCWriteCond para a escrita no registrador PC do valor em Alu.

Estado 12(bne):

Neste estado realizamos a comparação de não igual do valor no registrador A com o valor no registrador B e setamos os valores de PCWriteCondBne para a escrita no registrador PC do valor em Alu.

Estado 13(lui):

Neste estado realizamos a estensão do Imm e escrevemos esse valor no registrador destino.

Estado 14(and):

Neste estado ocorre a operação de and do valor no registrador A com o valor no registrador B e o valor é guardado no registrador AluOut e estará disponível no próximo estado.

Estado 15(slt):

Neste estado ocorre a subtração do valor no registrador A com o valor no registrador B e o bit 63 desta operação é escrito no registrador destino. Além disso ocorre verificação de OverFlow.

Estado 16(slti):

Neste estado ocorre a subtração do valor no registrador A com o Imm estendido e o bit 63 desta operação é escrito no registrador destino. Além disso ocorre verificação de OverFlow.

Estado 17(jalr):

Neste estado ocorre a escrita de PC no registrador destino.

Estado 18:

Neste estado calculamos o novo valor de PC pela soma do valor no registrador A com o Imm estendido e esse valor é escrito em AluOut, estará disponível no próximo estado.

Estado 19(srli):

Neste estado passamos o valor do registrador A pelo módulo shift2, que realiza o shift right lógico shamt vezes, e a saída é escrita no registrador destino.

Estado 20(srai):

Neste estado passamos o valor do registrador A pelo módulo shift2, que realiza o shift right aritmético shamt vezes, e a saída é escrita no registrador destino.

Estado 21(slli):

Neste estado passamos o valor do registrador A pelo módulo shift2, que realiza o shift left lógico shamt vezes, e a saída é escrita no registrador destino.

Estado 22(jal):

Neste estado escrevemos o valor de PC no registrador destino.

Estado 23:

Neste estado calculamos o novo valor de PC pela soma do PC com o Imm estendido e multiplicado por 4, esse resultado é escrito em AluOut e estará disponível no estado seguinte.Além disso, confere se ocorreu OverFlow.

Estado 24(bge):

Neste estado realizamos a comparação de maior igual do valor no registrador A com o valor no registrador B e setamos os valores de PCWriteCondBge para a escrita no registrador PC do valor em Alu. Além disso, confere se ocorreu OverFlow.

Estado 25(blt):

Neste estado realizamos a comparação de menor do valor no registrador A com o valor no registrador B e setamos os valores de PCWriteCondBlt para a escrita no registrador PC do valor em Alu. Além disso, confere se ocorreu OverFlow.

Estado 26(lb):

Neste estado escrevemos a saída do módulo historia, que contém o byte a ser carregado estendido, e escrevemos no registrador destino.

Estado 27(lh):

Neste estado escrevemos a saída do módulo historia, que contém a meia palavra ser carregada estendida, e escrevemos no registrador destino.

Estado 28(lw):

Neste estado escrevemos a saída do módulo historia, que contém a palavra a ser carregada estendida, e escrevemos no registrador destino.

Estado 29(lbu):

Neste estado escrevemos a saída do módulo historia, que contém o byte a ser carregado estendido com 0, e escrevemos no registrador destino.

Estado 30(lhu):

Neste estado escrevemos a saída do módulo historia, que contém a meia palavra a ser carregada estendido com 0, e escrevemos no registrador destino.

Estado 31(lwu):

Neste estado escrevemos a saída do módulo historia, que contém a palavra a ser carregada estendido com 0, e escrevemos no registrador destino.

Estado 32(reset):

Estado de inicialização da CPU em que desse estado vamos para o estado 0 onde ocorre o reset.

Estado 33:

Neste estado ocorre o cálculo do endereço, a partir da soma do valor no registrador A com o Imm estendido e o valor é guardado no registrador AluOut e estará disponível no próximo estado. Além disso, temos a verificação de OverFlow.

Estado 34:

Neste estado carregamos o registrador MDR a partir da saída da memória e verificamos qual o tipo de store será feito, sw, sh ou sb a partir dos bits de 14 a 12 da instrução.

Estado 35(sw):

Neste estado escrevemos na memória a saída do módulo armário, que contém a palavra a ser escrita concatenada com o que já havia na memória, a partir do endereço armazenado em AluOut.

Estado 36(sh):

Neste estado escrevemos na memória a saída do módulo armário, que contém a meia palavra a ser escrita concatenada com o que já havia na memória, a partir do endereço armazenado em AluOut.

Estado 37(sb):

Neste estado escrevemos na memória a saída do módulo armário, que contém o byte a ser escrita concatenado com o que já havia na memória, a partir do endereço armazenado em AluOut.

Estado 40(break):

Neste estado não fazemos nada apenas mantemos o código num loop infinito

Estado 45(OPCODE inexistente):

Neste estado escrevemos o valor de PC da instrução atual no registrador EPC e solicitamos leitura da memória a partir do endereço 254 e guardamos o byte lido no registrador NewPC.

Estado 46:

Neste estado escrevemos em PC o valor armazenado no registrador NewPC.

Estado 47(Overflow):

Neste estado escrevemos o valor de PC da instrução atual no registrador EPC e solicitamos leitura da memória a partir do endereço 255 e guardamos o byte lido no registrador NewPC.

Estado 48:

Neste estado ocorre a escrita do valor armazenado em AluOut no registrador destino.

Estado 51:

Neste estado ocorre a solicitação de leitura da memória a partir do endereço armazenado em AluOut.

Estado 58:

Neste estado ocorre a escrita no registrador PC do valor armazenado em Alu.

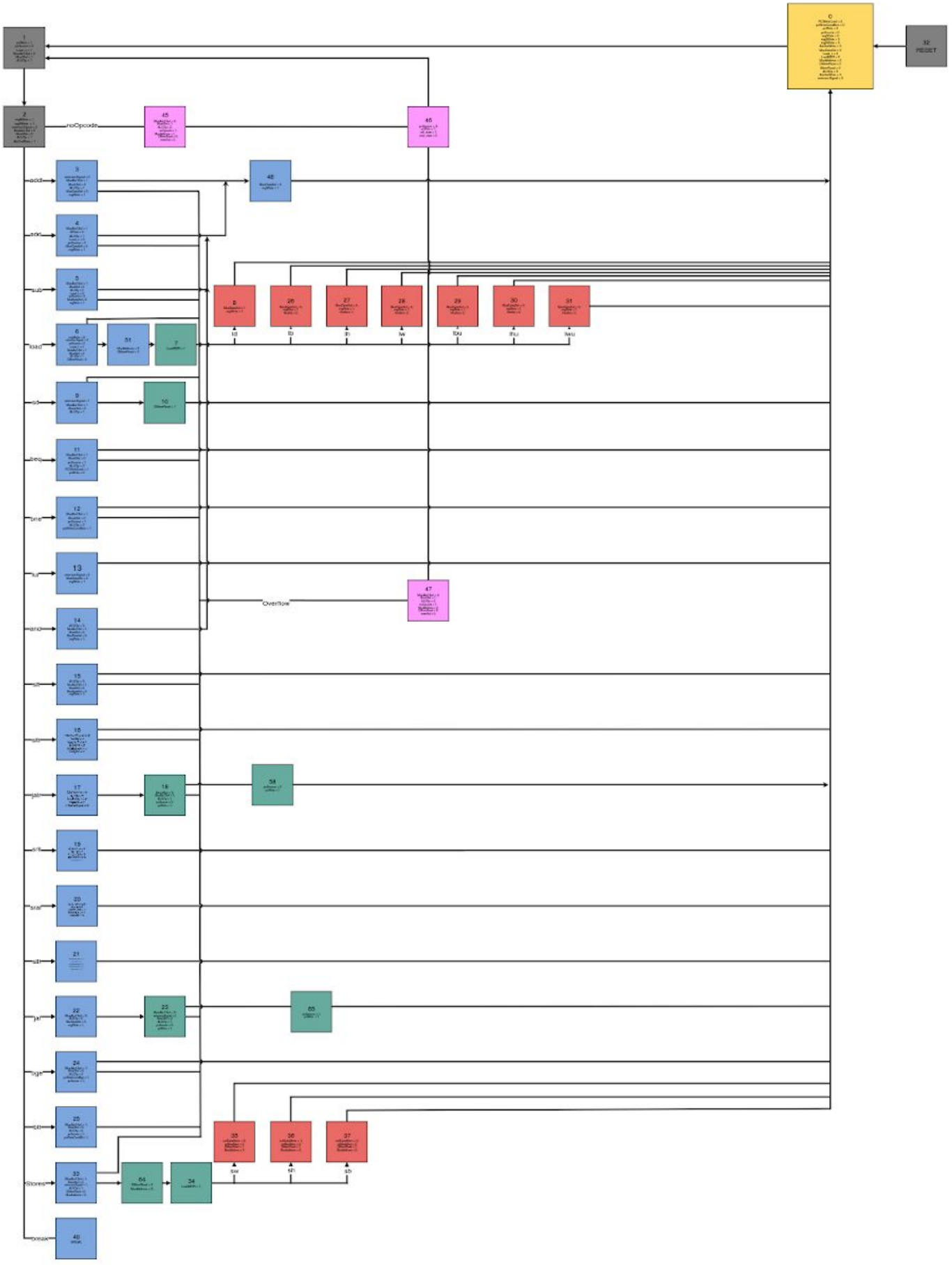
Estado 64:

Neste estado ocorre a solicitação de leitura da memória a partir do endereço armazenado em AluOut.

Estado 65:

Neste estado ocorre a escrita no registrador PC do valor armazenado em AluOut.

MÁQUINA DE ESTADOS DA UNIDADE DE PROCESSAMENTO



ANEXOS

Máquina de estados:
https://github.com/GabrielOlem/ProjetoHardware/blob/master/Montador_Execut%C3%A1vel/fsm.pdf

Unidade de controle:
https://github.com/GabrielOlem/ProjetoHardware/blob/master/cpu.jpg