Universidade Federal de Pernambuco
Centro de Informática

Relatório de Projeto

Felipe Nunes de Almeida Pereira (fnap)
Marcelo Victor Batista da Silva (mvbs3)
Maria Luísa Leandro de Lima (mlll)
Thiago da Silva Araújo (tsa3)

Professora Edna Natividade da Silva Barros

Recife, 11 de outubro de 2019

Índice

1. Descrição dos módulos
2. Descrição das operações
    2.1. Instruções do tipo R
    2.2. Instruções do tipo I
    2.3. Instruções do tipo S
    2.4. Instruções do tipo SB
    2.5. Instruções do tipo U
    2.6. Instruções do tipo UJ
3. Descrição dos estados de controle
4. Máquina de estados da Unidade de Controle

1. Descrição dos Módulos

1.1 Módulo: myData

Entradas

entrada1 (64 bits): vetor de bits a ser em parte concatenado com a entrada2.

entrada2 (64bits): vetor de bits a ser em parte concatenado com a entrada1.

qualTipo (4 bits): seletor que seleciona o tipo de concatenação a ser realizada.

Saída

concatenado (64 bits): vetor de bits concatenado de acordo com o seletor fornecido.

Objetivo

Módulo combinacional construído para realizar loads e stores de words, halfwords e bytes sem a perda de dados decorrente de uma sobrescrita. A concatenação que o módulo fornece provê um método de sobrescrever apenas o objetivado, salvando os dados que não se deve perder.

Algoritmo

O seletor qualTipo (modificado pela Unidade de Controle) designa o tipo de concatenação a ser realizada. Esta ocorrerá de forma que a entrada1 fornece os 8, 16 ou 32 primeiros bits do concatenado e a entrada2 fornece os restantes (a diferença entre os bits fornecidos pela entrada1 e 64).

1.2 Módulo:meuCpu(uniCpu)

Entradas

Clk(1 bit): São os pulsos que determinam quando o módulo vai operar.

Reset (1 bit): É o bit que reseta todos os módulos.

Saídas

Não há saídas.

Objetivo

Este módulo serve para juntar todos os outro módulos, basicamente todos os outros módulos são “instanciados nesse módulo”, as entradas da unidade de controle vêm desse módulo. As saídas da unidade de controle vão para esse módulo. Sendo assim, esse módulo serve como uma interface de todos os módulos entre todos os módulos e da unidade de controle com todos os outros módulos

1.3 Módulo: SignExtend(meuExtensor)

Entradas

instrução(32 bits): Instrução que contém o imediato a ser calculado.

indicaImmediate(4 bits): seletor que indica o formato do imediato dentro da instrução, a fim de formatá-lo corretamente para a operação a ser realizada com ele.

Saída

immediate(64 bits): imediato formatado e deslocado para a operação para a qual ele foi formatado.

Objetivo

Módulo combinacional construído para formatar o imediato que vem em uma instrução, deixando-o pronto para o uso na operação que o for utilizar. Este módulo realiza a concatenação das partes que estiverem separadas dentro da instrução e o deslocamento que for necessário.

Algoritmo

O seletor indicaImmediate (modificado pela Unidade de Controle) designa o tipo de instrução a ser realizada, que indica as posições do imediato dentro da instrução. Sabendo este formato, realiza-se a concatenação adequada e realiza-se o deslocamento a esquerda. O resultado destas operações é a saída immediate. A partir do valor do indicaImmediate temos:

1-Immediates de funções do tipo I

2-Immediates de funções do tipo SB

3-Immediates de funções do tipo U

4-Immediates de funções do tipo S

5-Immediates de funções do tipo UJ

1.4 Módulo: UniControle(uniCpu)

Entradas

clk(1 bit): bit que indica quando o módulo irá operar.

rst_n(1 bit) bit que reinicia todos os módulos.

instrucao(32 bits): Saída do registrador de instrução.

opcode(7 bits): Indica o tipo de instrução.

iguais(1 bits): bit da Ula que indica que as duas entradas têm o mesmo valor.

menor(1 bit): bit da Ula que indica que a entrada A da Ula é menor que a entrada B.

maior(1 bit): bit da Ula que indica que a entrada A da Ula é maior que a entrada B.

Overflow(1 bit): bit da Ula que indica que ocorreu o overflow em uma operação aritmétrica.

Saída

estadoUla(3 bits): Indica qual operação a ula deve realizar.

escritaPC(1 bit): habilita escrita no registrador PC.

RWmemoria(1 bit): Seleciona leitura ou escrita na memória de intruções

escreveInstr(1 bit): habilita escrita no registrador de instruções

escreveA(1 bit) :habilita escrita no registrador A.

escreveB:(1 bit): habilita escrita no registrador B.

escreveALUOut(1 bit): habilita escrita no registrador RegAluOut

escreveNoBancoReg(1 bit): habilita escrita no banco de registradores.

SeletorMuxA(4 bits): seleciona a saída do mux A.

SeletorMuxB(4 bits): seleciona a saída do mux B.

SeletorMuxW(4 bits): seleciona a saída do MuxWrite(mux de escrita).

indicaImmediate(4 bits): indica como calcular o immediate para o módulo signExtend.

SeletorMuxPC(4 bits): seleciona a saída do muxPc

LerEscreMem64(1 bit): escolhe entre leitura e escrita da memoria64

selShift(2 bits): seleciona qual o tipo de shift.

seletorMuxMem64(4 bits): seleciona qual valor vai entrar na memoria64

regEscreveMDR(1 bit): habilita escrita no registrador MDR

seletorMeuDado(4 bits): seleciona a operação do modulo meuDado

escreveEPC(1 bit): habilita escrita no registrador EPC

escreveEPCProvisorio(1 bit): habilita escrita no registrador EPCprovisorio

Objetivo

Este módulo basicamente seleciona em qual estado a CPU se encontra no momento e também serve para mandar sinais de controle para todos os módulos.

Algoritmo

O algoritmo executa um estado e atualiza os sinais de controle. Em seguida checa as entradas. Baseado em tais entradas ele determina qual o próximo estado do processo(ver máquina de estados).

2. Descrição das operações

2.1.Instruções do tipo R

- Instrução add

Formato: add rd, rs1, rs2

Descrição:

A instrução é decodificada e, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B, respectivamente, seus valores são levados à ALU para a operação de soma, que é selecionada por um sinal da Unidade de Controle, a saída do ALU é escrita no registrador regAluOut. Depois, o resultado é armazenado em no banco de registradores na posição rd e uma nova instrução é buscada.

-Instrução sub

Formato: sub rd, rs1, rs2

A instrução é decodificada e, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B, respectivamente, seus valores são levados à ALU para a operação de subtração, que é selecionada por um sinal da Unidade de Controle, a saída do

ALU é escrita no registrador regAluOut. Depois, o resultado é escrito no banco de registradores na posição rd e uma nova instrução é buscada.

-Instrução and

Formato: and rd,rs1,rs2

A instrução é decodificada e, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B, seus valores são levados à ALU para a operação and, que é selecionada por um sinal da Unidade de Controle, a saída do ALU é escrita no registrador regAluOut . Depois, o resultado é armazenado em no banco de registradores na posição rd e uma nova instrução é buscada.

-Instrução slt

Formato: slt rd, rs1, rs2

A instrução é decodificada e, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B, seus valores são levados à ALU para a operação de comparação menor, que é selecionada por um sinal da Unidade de Controle. Depois, se a comparação for verdadeira na posição do registrador rd do banco de registradores será carregado o valor 1, caso contrário será carregado o valor 0 e uma nova instrução é buscada.

2.2. Instruções do tipo I

-Instrução addi

Formato: addi rd, rs1, imm

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A e seu valor é levado à ALU, a partir de um sinal da unidade de controle (indicaImmediate), o módulo SignExtend calcula o valor do imediato e o leva para a ALU para a operação de soma, que é selecionada por um sinal da Unidade de Controle, a saída do ALU é escrita no registrador regAluOut. Depois, o resultado é armazenado no banco de registradores na posição rd e uma nova instrução é buscada.

-Instrução slti

Formato: slti rd, rs1, imm

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A e seu valor é levado a ALU, a partir de um sinal da unidade de controle (indicaImmediate), o módulo SignExtend calcula o valor do imediato e o leva para Alu para a operação de comparação menor, que é selecionada por um sinal da Unidade de Controle. Depois, se a comparação for verdadeira o banco de registradores vai carregar na posição do registrador rd o valor 1, caso contrário rd receberá 0 e uma nova instrução é buscada.

-Instrução jalr

Formato: jalr rd,rs1,imm

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A e seu valor é levado a ALU. A partir de um sinal da unidade de controle (indicaImmediate), o módulo SignExtend calcula o valor do imediato e o leva para ALU para a operação de soma, que é selecionada por um sinal da Unidade de Controle. A saída do ALU é escrita no registrador regAluOut. Depois, o PC receberá o valor armazenado no regAluOut enquanto no banco de registradores é carregado o valor atual de PC(antes da modificação) no endereço de rd e uma nova instrução é buscada.

-Instrução lb

Formato: lb rd,imm(rs1)

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A e seu valor é levado a ALU, a partir de um sinal da unidade de controle (indicaImmediate), o módulo SignExtend calcula o valor do imediato e o leva para ALU para a operação de soma, que é selecionada por um sinal da Unidade de Controle, a saída do ALU é escrita no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64 e o valor que tal endereço contém é armazenado no MDR (registrador de saída da Memória 64), o valor da saída do MDR é colocado na entrada do módulo meuDado, através de um sinal da unidade de controle (qualTipo), o módulo carrega no banco de registradores no registrador rd o valor dos 8 bits menos significativos com o sinal estendido (para 64bits) que tal endereço contém e uma nova instrução é buscada.

-Instrução lh

Formato: lh rd,imm(rs1)

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A e seu valor é levado a ALU, a partir de um sinal da unidade de controle (indicaImmediate), o módulo SignExtend calcula o valor do imediato e o leva para ALU para a operação de soma, que é selecionada por um sinal da Unidade de Controle, a saída do ALU é escrita no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64 e o valor que tal endereço contém é armazenado no MDR (registrador de saída da Memória 64), o valor da saída do MDR é colocado na entrada do módulo meuDado, através de um sinal da unidade de controle (qualTipo), o módulo carrega no banco de registradores no registrador rd o valor dos 16 bits menos significativos com o sinal estendido (para 64bits) que tal endereço contém e uma nova instrução é buscada.

-Instrução lw

Formato: lw rd,imm(rs1)

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A e seu valor é levado a ALU, a partir de um sinal da unidade de controle (indicaImmediate), o módulo SignExtend calcula o valor do imediato e o leva para ALU para a operação de soma, que é selecionada por um sinal da Unidade de Controle, a saída do ALU é escrita no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64 e o valor que tal endereço contém é armazenado no MDR (registrador de saída da Memória 64), o valor da saída do MDR é colocado na entrada do módulo meuDado, através de um sinal da unidade de controle (qualTipo), o módulo carrega no banco de registradores no registrador rd o valor dos 32 bits menos significativos com o sinal estendido (para 64bits) que tal endereço contém e uma nova instrução é buscada.

-Instrução ld

Formato: ld rd, imm(rs1)

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A. O imediato é concatenado no módulo SignExtend e, junto com o valor de rs1, é levado para a ALU onde a operação de soma é feita, sendo selecionada por um sinal da Unidade de Controle, a saída do ALU é escrita no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64 e o valor que tal endereço contém é armazenado no

MDR (registrador de saída da Memória 64) e em seguida no banco de registradores no endereço do registrador rd e uma nova instrução é buscada.

-Instrução lbu

Formato: lbu rd,imm(rs1)

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A e seu valor é levado a ALU, a partir de um sinal da unidade de controle (indicaImmediate), o módulo SignExtend calcula o valor do imediato e o leva para ALU para a operação de soma, que é selecionada por um sinal da Unidade de Controle, a saída do ALU é escrita no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64 e o valor que tal endereço contém é armazenado no MDR (registrador de saída da Memória 64), o valor da saída do MDR é colocado na entrada do módulo meuDado, através de um sinal da unidade de controle (qualTipo), o módulo carrega no banco de registradores no registrador rd o valor dos 8 bits menos significativos é estendido com zeros (para 64bits) que tal endereço contém e uma nova instrução é buscada.

-Instrução lhu

Formato: lhu rd,imm(rs1)

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A e seu valor é levado a ALU, a partir de um sinal da unidade de controle (indicaImmediate), o módulo SignExtend calcula o valor do imediato e o leva para ALU para a operação de soma, que é selecionada por um sinal da Unidade de Controle, a saída do ALU é escrita no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64 e o valor que tal endereço contém é armazenado no MDR (registrador de saída da Memória 64), o valor da saída do MDR é colocado na entrada do módulo meuDado, através de um sinal da unidade de controle (qualTipo), o módulo carrega no banco de registradores no registrador rd o valor dos 16 bits menos significativos é estendido com zeros (para 64bits) que tal endereço contém e uma nova instrução é buscada.

-Instrução lwu

Formato: lwu rd,imm(rs1)

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A e seu valor é levado a ALU, a partir de um sinal da unidade de controle (indicaImmediate), o módulo SignExtend calcula o valor do imediato e o leva para ALU para a operação de soma, que é selecionada por um sinal da Unidade de Controle, a saída do ALU é escrita no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64 e o valor que tal endereço contém é armazenado no MDR (registrador de saída da Memória 64), o valor da saída do MDR é colocado na entrada do módulo meuDado, através de um sinal da unidade de controle (qualTipo), o módulo carrega no banco de registradores no registrador rd o valor dos 32 bits menos significativos é estendido com zeros (para 64bits) que tal endereço contém e uma nova instrução é buscada.

-Instrução nop

Formato: nop

É uma instrução que não faz nada.

-Instrução break

Formato: break

É uma instrução que não faz nada e que possui como próximo estado ela mesma (trava o programa).

-Instrução srli

Formato: srli rd,rs1,shamt

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A. O valor de A vai para o módulo MeuShift, onde ocorrerá o shift lógico (completando com zeros) para a direita a partir do valor do shamt (funct 6). Depois, o resultado é armazenado no banco de registradores na posição rd.

-Instrução srai

Formato: srai rd,rs1,shamt

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A. O valor de A vai para o módulo MeuShift, onde ocorrerá o shift aritmético

(com extensão de sinal) para a direita a partir do valor do shamt (funct 6). Depois, o resultado é armazenado no banco de registradores na posição rd.

-Instrução slli

Formato: slli rd,rs1,shamt

A instrução é decodificada e, após isso, rs1 é carregado pelo banco de registradores no registrador A. O valor de A vai para o módulo MeuShift, onde ocorrerá o shift lógico (completando com zeros) para a esquerda a partir do valor do shamt (funct 6). Depois, o resultado é armazenado no banco de registradores na posição rd.

2.3 Instruções do tipo S

-Instrução sd

Formato: sd rs2, imm(rs1)

A instrução é decodificada e, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B. O imediato é concatenado no módulo SignExtend e, junto com o valor de rs1, é levado para a ALU onde a operação de soma é feita, sendo selecionada por um sinal da Unidade de Controle. O valor de saída da ALU é colocado no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64 e o valor de rs2 é escrito nesse endereço de memória e uma nova instrução é buscada.

-Instrução sw

Formato: sw rs2,imm(rs1)

A instrução é decodificada e, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B. O imediato é concatenado no módulo SignExtend e, junto com o valor de rs1, é levado para a ALU onde a operação de soma é feita, sendo selecionada por um sinal da Unidade de Controle. O valor de saída da ALU é colocado no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64, o valor lido neste endereço de memória é salvo no registrador MDR. Os 32 bits mais significativos do MDR é concatenado com os 32 bits menos significativos do registrador B (rs2) no módulo meuDado. A saída do módulo meuDado é escrita na memória no endereço regAluOut.

-Instrução sh

Formato: sh rs2,imm(rs1)

A instrução é decodificada e, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B. O imediato é concatenado no módulo SignExtend e, junto com o valor de rs1, é levado para a ALU onde a operação de soma é feita, sendo selecionada por um sinal da Unidade de Controle. O valor de saída da ALU é colocado no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64, o valor lido neste endereço de memória é salvo no registrador MDR. Os 48 bits mais significativos do MDR é concatenado com os 16 bits menos significativos do registrador B (rs2) no módulo meuDado. A saída do módulo meuDado é escrita na memória no endereço regAluOut.

-Instrução sb

Formato: sb rs2,imm(rs1)

A instrução é decodificada e, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B. O imediato é concatenado no módulo SignExtend e, junto com o valor de rs1, é levado para a ALU onde a operação de soma é feita, sendo selecionada por um sinal da Unidade de Controle. O valor de saída da ALU é colocado no registrador regAluOut. Depois, o resultado é usado como endereço na Memória64, o valor lido neste endereço de memória é salvo no registrador MDR. Os 56 bits mais significativos do MDR é concatenado com os 8 bits menos significativos do registrador B (rs2) no módulo meuDado. A saída do módulo meuDado é escrita na memória no endereço regAluOut.

2.4 Instruções do tipo SB

-Instrução beq

Formato: beq rs1,rs2,imm

A instrução é decodificada, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B e imediato é concatenado no módulo SignExtend, o valor do PC e o imediato são levados para a ULA, é realizada a operação de soma e o resultado é carregado no registrador RegAluOut. Em seguida os valores de rs1 e rs2 são levados à ALU para a operação de comparação, que é selecionada por um sinal da Unidade de Controle. Caso os registradores possuam o mesmo valor, o SeletorMuxPc terá o sinal 1 e o MuxPc vai enviar o valor do regAluOut para o PC, uma nova instrução é buscada.

-Instrução bne

Formato: bne rs1,rs2,imm

A instrução é decodificada, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B e imediato é concatenado no módulo SignExtend, o valor do PC e o imediato são levados para a ULA, é realizada a operação de soma e o resultado é carregado no registrador RegAluOut. Em seguida os valores de rs1 e rs2 são levados à ALU para a operação de comparação, que é selecionada por um sinal da Unidade de Controle. Caso os registradores possuam valores diferentes, o SeletorMuxPc terá o sinal 1 e o MuxPc vai enviar o valor do regAluOut para o PC, uma nova instrução é buscada.

-Instrução bge

Formato: bge rs1,rs2,imm

A instrução é decodificada, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B e imediato é concatenado no módulo SignExtend, o valor do PC e o imediato são levados para a ULA, é realizada a operação de soma e o resultado é carregado no registrador RegAluOut. Em seguida os valores de rs1 e rs2 são levados à ALU para a operação de comparação, que é selecionada por um sinal da Unidade de Controle. Caso o valor do registrador A for maior ou igual que o do registrador B, o SeletorMuxPc terá o sinal 1 e o MuxPc vai enviar o valor do regAluOut para o PC, uma nova instrução é buscada.

-Instrução blt

Formato: blt rs1,rs2,imm

A instrução é decodificada, após isso, rs1 e rs2 são carregados pelo banco de registradores nos registradores A e B e imediato é concatenado no módulo SignExtend, o valor do PC e o imediato são levados para a ULA, é realizada a operação de soma e o resultado é carregado no registrador RegAluOut. Em seguida os valores de rs1 e rs2 são levados à ALU para a operação de comparação, que é selecionada por um sinal da Unidade de Controle. Caso o valor do registrador A for menor que o do registrador B, o SeletorMuxPc terá o sinal 1 e o MuxPc vai enviar o valor do regAluOut para o PC, uma nova instrução é buscada.

2.5 Instruções do tipo U

-Instrução lui

Formato: lui rd, imm

A instrução é decodificada e, após isso, o imediato é "shiftado" para a esquerda 12 vezes. Depois, será armazenado o valor do imediato no banco de registradores na posição rd e uma nova instrução é buscada.

2.6 Instruções do tipo UJ

-Instrução jal

Formato: jal rd,imm

A instrução é decodificada. O valor de PC é levevado para a ULA. A partir de um sinal da unidade de controle (indicaImmediate), o módulo SignExtend calcula o valor do imediato e o leva para ALU para a operação de soma, que é selecionada por um sinal da Unidade de Controle. A saída do ALU é escrita no registrador regAluOut. Depois, o PC receberá o valor armazenado no regAluOut enquanto no banco de registradores é carregado o valor atual de PC(antes da modificação) no endereço de rd e uma nova instrução é buscada.

3. Descrição dos estados de controle

3.1. reset

Atribui o valor zero a todas as variáveis de controle da unidade e atribui o próximo estado para addPC(3.2).

3.2. addPC

Neste estado a ULA realiza a operação 'PC + 4' e atribui o próximo estado como testaOpcode(3.21) e salva o valor de PC no registrador EPCProvisório.

3.3. esperaMeuDado

Neste estado, é necessário esperar a operação do estado pegaEConcatena(3.5) terminar, que escreve um dado no MDR, levando um ciclo. esperaMeuDado aguarda até que a informação seja escrita em MDR para utilizá-la.

3.4. **esperaOMdr**

Este estado apenas ocupa um pulso de clock para que se possa ter a informação que vem da memória.

3.5. **pegaEConcatena**

Este estado é executado quando há uma operação de load ou store. Se for um load, ele acessa o que está no banco de registradores e escreve esta informação. Se for um store, ele acessa a memória e pula para um estado que espera a informação ser recebida.

3.6. **escreveDadoBancoReg**

Utiliza o dado contido no MDR e o armazena no banco de registradores.

3.7. **escrita**

Ativa o sinal de escrita no banco de registradores para que se possa escrever as informações calculadas e põe o próximo estado como addPC(3.2) para que se continue o fluxo do programa.

3.8. **sb**

Neste estado é inicializado o início de uma sequência de estados que irão passarão no pegaEConcatena(3.5) e esperaOMdr(3.4), para armazenar o valor de 1 byte no rd da instrução.

3.9. **sh**

Neste estado é inicializado o início de uma sequência de estados que irão passarão no pegaEConcatena(3.5) e esperaOMdr(3.4), para armazenar o valor de 2 bytes no rd da instrução.

3.10. **sw**

Neste estado é inicializado o início de uma sequência de estados que irão passarão no pegaEConcatena(3.5) e esperaOMdr(3.4), para armazenar o valor de 4 bytes no rd da instrução.

3.11.**lb**

Neste estado é inicializado o processo de escrever no banco de registradores dados de 1 byte, que passa pelo pegaEConcatena(3.5) e escreveDadoBancoReg(3.6).

3.12.**lh**

Neste estado é inicializado o processo de escrever no banco de registradores dados de 2 bytes, que passa pelo pegaEConcatena(3.5) e escreveDadoBancoReg(3.6).

3.13.**lw**

Neste estado é inicializado o processo de escrever no banco de registradores dados de 4 bytes, que passa pelo pegaEConcatena(3.5) e escreveDadoBancoReg(3.6).

3.14.**lbu**

Neste estado é inicializado o processo de escrever no banco de registradores dados de 1 byte sem sinal, que passa pelo pegaEConcatena(3.5) e escreveDadoBancoReg(3.6). Em myData(1.1), o bit mais significativo é colocado com 0 para que o sinal nunca seja negativo.

3.15.**lhu**

Neste estado é inicializado o processo de escrever no banco de registradores dados de 2 bytes sem sinal, que passa pelo pegaEConcatena(3.5) e escreveDadoBancoReg(3.6). Em myData(1.1), o bit mais significativo é colocado com 0 para que o sinal nunca seja negativo.

3.16.**lwu**

Neste estado é inicializado o processo de escrever no banco de registradores dados de 4 bytes sem sinal, que passa pelo pegaEConcatena(3.5) e escreveDadoBancoReg(3.6). Em myData(1.1), o bit mais significativo é colocado com 0 para que o sinal nunca seja negativo.

3.17.**andS**

É posto o estado nativo da ULA para que se faça a comparação **E** entre os dois valores dos registradores.

3.18. **jalr**

Põe a ULA no estado de soma para calcular o offset do pulo e continua para o próximo estado esperaJalr(3.19).

3.19. **esperaJalr**

Este estado espera um ciclo de clock que é o tempo necessário que o offset + rs1 seja calculado e posto no AluOut para que seja escrita no próximo ciclo: escritaPulos(3.20).

3.20. **escritaPulos**

Com o dado já calculado, ele é guardado no banco de registradores(rd).

3.21. **slli**

Ao identificar que é um pulo na decodificação da instrução(3.24) põe como o próximo estado a escrita(3.7) do resultado, habilitando uma flag chamada escrevemeushift. Resultado este que já foi computado no módulo meuShift(1.4) de acordo com seu respectivo shift.

3.22. **srli**

Ao identificar que é um pulo na decodificação da instrução(3.24) põe como o próximo estado a escrita(3.7) do resultado, habilitando uma flag chamada escrevemeushift. Resultado este que já foi computado no módulo meuShift(1.4) de acordo com seu respectivo shift.

3.23. **srai**

Ao identificar que é um pulo na decodificação da instrução(3.24) põe como o próximo estado a escrita(3.7) do resultado, habilitando uma flag chamada escrevemeushift. Resultado este que já foi computado no módulo meuShift(1.4) de acordo com seu respectivo shift.

3.24. **testaOpcode**

Este é um estado muito importante pois algumas tarefas que devem ser feitas em outros estados são iniciadas aqui, a partir de vários testes de condição dentro do caso testaOpcode. Feita a análise do opcode (e/ou funct3 e/ou funct6) decidindo qual estado deve se suceder, ele já altera algumas variáveis que iniciam o processo de, por exemplo, cálculo de

pulos, verificações de condições de branches condicionais e parâmetros para os vários tipos de shifts.

3.25. **breaks**

Estado onde seu próximo estado é ele mesmo.

3.26. **bge**

Leva em consideração uma variável chamada: menor, que guarda o resultado da ULA da comparação entre os dois registradores fonte, se o rs1 for, de fato, menor que o rs2, esta variável terá o valor de true. Partindo disso, se faz uma comparação: Se “menor” é verdadeiro, continua-se para pc+4. Caso seja falsa, o próximo estado é direcionado para escritaPulos(3.38).

3.27. **blt**

Leva em consideração uma variável chamada: menor, que guarda o resultado da ULA da comparação entre os dois registradores fonte, se o rs1 for, de fato, menor que o rs2, esta variável terá o valor de true. Partindo disso, se faz uma comparação: Se “menor” é verdadeiro, o próximo estado é direcionado para escritaPulos(3.38).Caso seja falso, continua-se para pc+4.

3.28. **add**

Modifica o estado da ULA para a realização da soma dos operandos e define o próximo estado como a escrita(3.39) do resultado calculado.

3.29. **sub**

Modifica o estado da ULA para a realização da subtração entre os operandos e define o próximo estado como a escrita(3.39) do resultado calculado.

3.30. **addi**

Modifica o estado da Ula para a realização da soma dos operandos e modifica o sinal que indica ao Módulo signExtend(1.3) o formato do imediato na instrução.

3.31. **beq**

Leva em consideração uma variável chamada: igual, que guarda o resultado da ULA da comparação entre os dois registradores fonte, se o rs1 for, de fato, igual ao rs2, esta variável terá o valor de true. Partindo disso, se faz uma comparação: Se “igual” é verdadeiro, o próximo estado é direcionado para escritaPulos(3.38).Caso seja falso, continua-se para addPc(3.2)..

3.32. **bne**

Leva em consideração uma variável chamada: igual, que guarda o resultado da ULA da comparação entre os dois registradores fonte, se o rs1 for, de fato, igual ao rs2, esta variável terá o valor de true. Partindo disso, se faz uma comparação: Se “igual” é verdadeiro, o próximo estado é direcionado para addPc(3.2).Caso seja falso, o fluxo é direcionado para escritaPulos(3.38).

3.33. **lui**

Modifica o sinal que indica ao Módulo meuExtensor(1.3) o formato do imediato na instrução. Ele é calculado e então é escrito no próximo estado: escrita(3.39).

3.34. **ld**

Habilita a leitura de uma doubleword(4 bytes) no endereço de memória designado na instrução.

3.35. **sd**

Habilita a escrita de uma doubleword(4 bytes) no endereço de memória designado na instrução.

3.36. **jal**

Modifica o estado da ULA para somar o endereço base e o imediato que fazem o jump. O jal não é responsável por designar o cálculo, isto é feito no testaOpcode(3.24). O estado que sucede é o escritaPulos(3.38).

.

3.37. **escritaMemoria**

Estado “buffer” para garantir a escrita na memória de 64 bits dentro da mesma instrução que a designou.

3.38. **nops**

Modifica o estado da Ula para a realização de uma soma “vazia” e modifica o sinal que indica ao Módulo signExtend(1.3) o formato do imediato na instrução, que neste caso, será 0.

3.39. **esperaLoad**

Estado “buffer” para garantir a escrita no registrador MDR dentro da mesma instrução que a designou.

3.40. **esperaLoadOutros**

Estado “buffer” para garantir a escrita no registrador MDR dentro da mesma instrução que a designou.

3.41. **slt**

Leva em consideração uma entrada chamada: menor, que guarda o resultado da ULA da comparação entre os dois registradores fonte, se o rs1 for, de fato, menor que o rs2, esta variável terá o valor de true. Partindo disso, se faz uma comparação: Se “menor” é verdadeiro, o valor que entra no banco de registradores a partir do MuxWrite é 1 e se for falso, o valor será 0.

3.42. **slti**

Leva em consideração uma entrada chamada: menor, que guarda o resultado da ULA da comparação entre um registrador fonte e o imediato, se o rs1 for, de fato, menor que o imediato, esta variável terá o valor de true. Partindo disso, se faz uma comparação: Se “menor” é verdadeiro, o valor que entra no banco de registradores a partir do MuxWrite é 1 e se for falso, o valor será 0.

3.43. **Inexistente**

Salva no registrador EPC o valor atual de PC e seleciona no MuxPC o valor de erro de opcodes inexistentes para que seja buscada na memória a rotina de tratamento para o erro no estado seguinte.

3.44. **esperaMem64**

Busca na memória a rotina de tratamento para o erro correspondente ao valor que foi inserido em PC anteriormente e a escreve no registrador MDR.

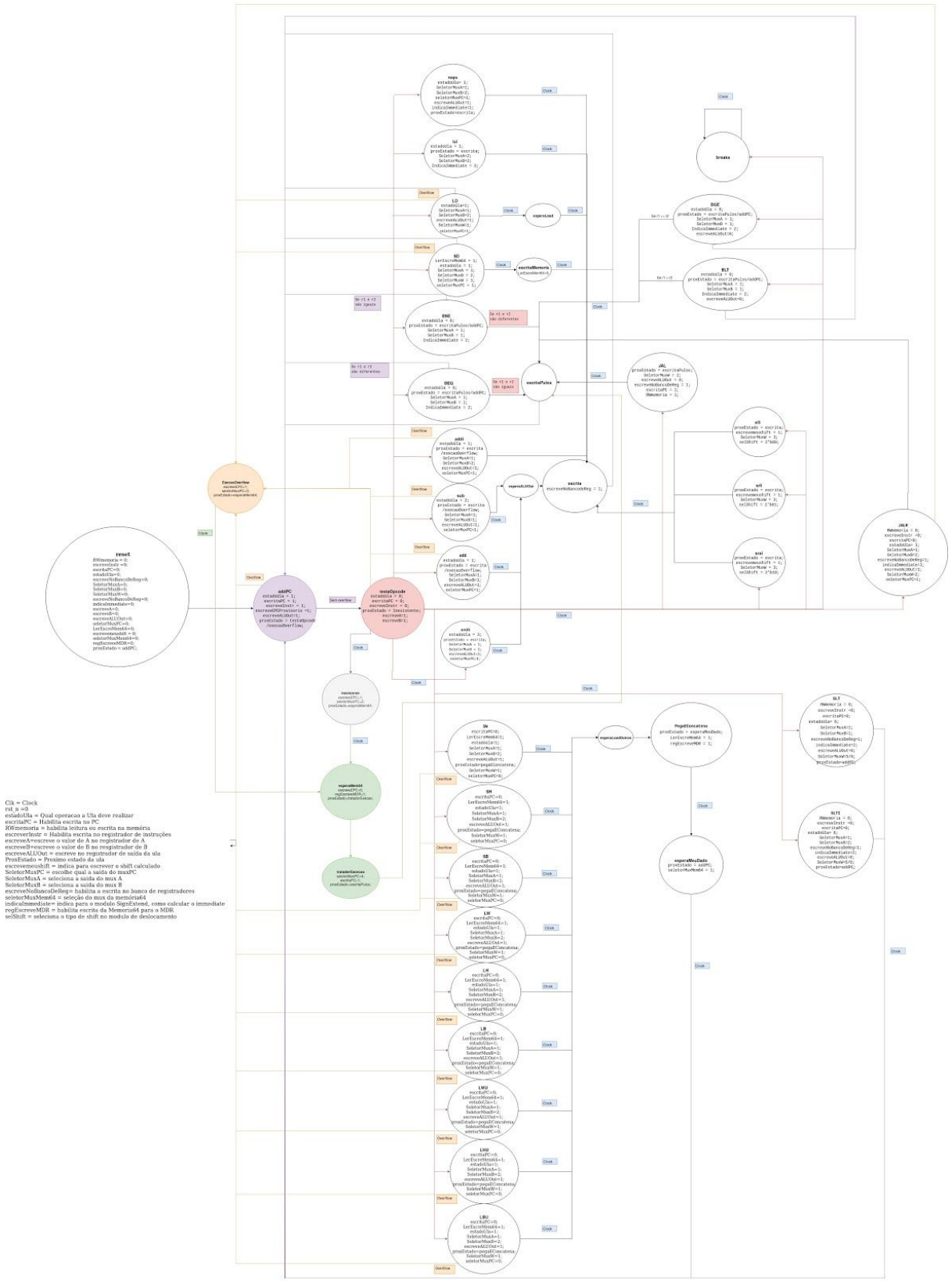
3.45. **tratadorExcecao**

Escreve em PC a saída do MDR para que a rotina de tratamento de exceções seja executada no próximo estado.

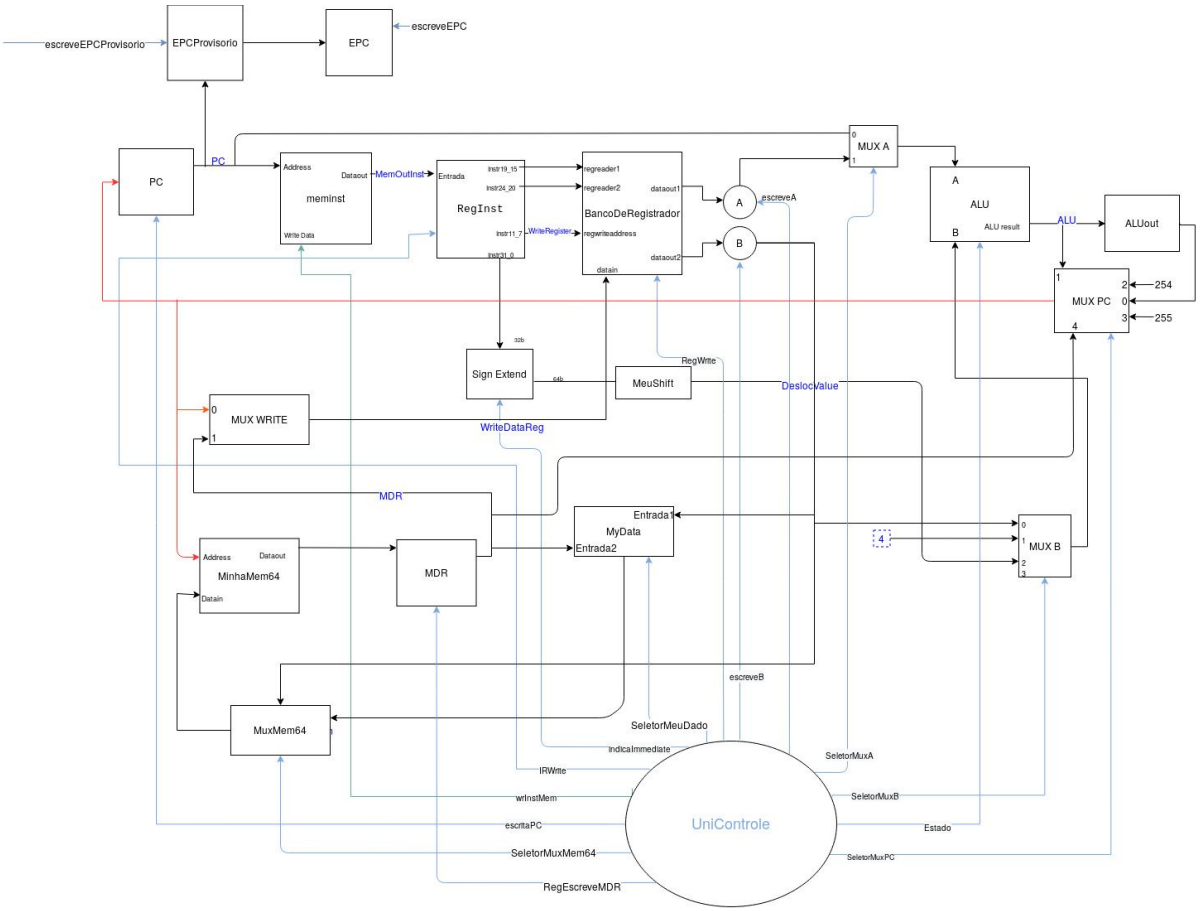
3.46. **ExecaoOverflow**

Salva no registrador EPC o valor atual de PC e seleciona no MuxPC o valor de erro de Overflow para que seja buscada na memória a rotina de tratamento para o erro no estado seguinte.

4. Máquina de estados da Unidade de Controle:



Datapath:


escreveEPCProvisorio
EPCProvisorio
EPC
escreveEPC
PC
PC
Address
Dataout
meminst
Write Data
MemOutInst
Entrada
RegInst
Instr19_15
Instr24_20
Instr11_7
Instr31_0
WriteRegister
regreader1
regreader2
dataout1
BancoDeRegistrador
regwriteaddress
dataout2
datain
A
B
escreveA
0
MUX A
1
A
ALU
B
ALU result
ALU
ALUout
1
2
254
MUX PC
0
3
255
4
32b
Sign Extend
64b
MeuShift
RegWrite
DeslocValue
0
MUX WRITE
1
WriteDataReg
MDR
Address
Dataout
MinhaMem64
Datain
MDR
Entrada1
MyData
Entrada2
4
0
1
MUX B
2
3
escreveB
MuxMem64
SeletorMeuDado
IndicaImmediate
IRWrite
wrInstMem
escritaPC
SeletorMuxMem64
RegEscreveMDR
UniControle
SeletorMuxA
SeletorMuxB
Estado
SeletorMuxPC