UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

RELATÓRIO DO PROJETO

Arthur Abrahão Santos Barbosa, AASB2

Gabriel Ferreira Rocha, GFR

José Lucas da Costa Silva, JLCS3

Lucas dos Reis Silva, LRS5

Recife, 11 de Outubro de 2019

Professora: Edna Natividade da Silva Barros

ÍNDICE

1. DESCRIÇÃO DOS MÓDULOS

1.1. Unidade de Controle (UC)

ENTRADAS:

rst (1 bit): Entrada de reset do sistema.

opcode (7 bits): Entrada para definir a função seguinte da instrução;

instruction (32 bits): Entrada para pegar o resto da instrução, usado para *funct3*, *funct6* e *funct7*.

SAÍDAS:

sel (3 bits): Define a operação da *Alu*;

pcwrite (1 bit): Determina se PC será escrito;

InstWrite (1 bit): Determina se o registrador de instruções será escrito;

RegWrite (1 bit): Determina se o banco de registradores será escrito;

menWrite(1 bit): Determina se a memória de dados será escrita;

MentoReg (1 bit): Define a saída do MUX da memória;

AluScrA (2 bits): Define a saída do MUX A;

AluScrB (2 bits): Defina a saída do MUX B;

loadAout (1 bit): Carrega o registrador ligado à *Alu*;

PCsource (1 bit): Define a saída do MUX do PC;

LoadregA (1 bit): Carrega o registrador A;

LoadregB (1 bit): Carrega o registrador B;

loadregMen (1 bit): Carrega o registrador da memória de dados;

Pcwritecond (3 bits): Em casos condicionais, checa pra ver se PC será escrito.

funct7 (7 bits): Define qual a função no mesmo tipo;

funct3 (3 bits): Define qual a função no mesmo tipo;

funct6 (6 bits): Define qual a função no mesmo tipo;

ShiftType (3 bits): Determina qual tipo de shift é feito em instruções tipo I (Shift).

OBJETIVO:

Regular o funcionamento do processador, manipulando quais informações serão escritas e quando, a partir da definição de estados dentro dele, permitindo assim a divisão das funções por meio deles.

ALGORITMO:

É um módulo sequencial baseado em máquina de estados, que sempre inicia no estado *"reset"*, onde todas suas saídas são definidas para 0, e depois passa para o estado *"sumPC"*, onde inicia a execução do código. A cada pulso de clock, o estado muda baseado no estado anterior e as entradas atuais, com as saídas em cada estado controlando a unidade de processamento.

1.2. Unidade de Processamento (UP)

ENTRADAS:

clk (1 bit): Representa o clock do sistema;

rst (1 bit): Entrada de reset do sistema.

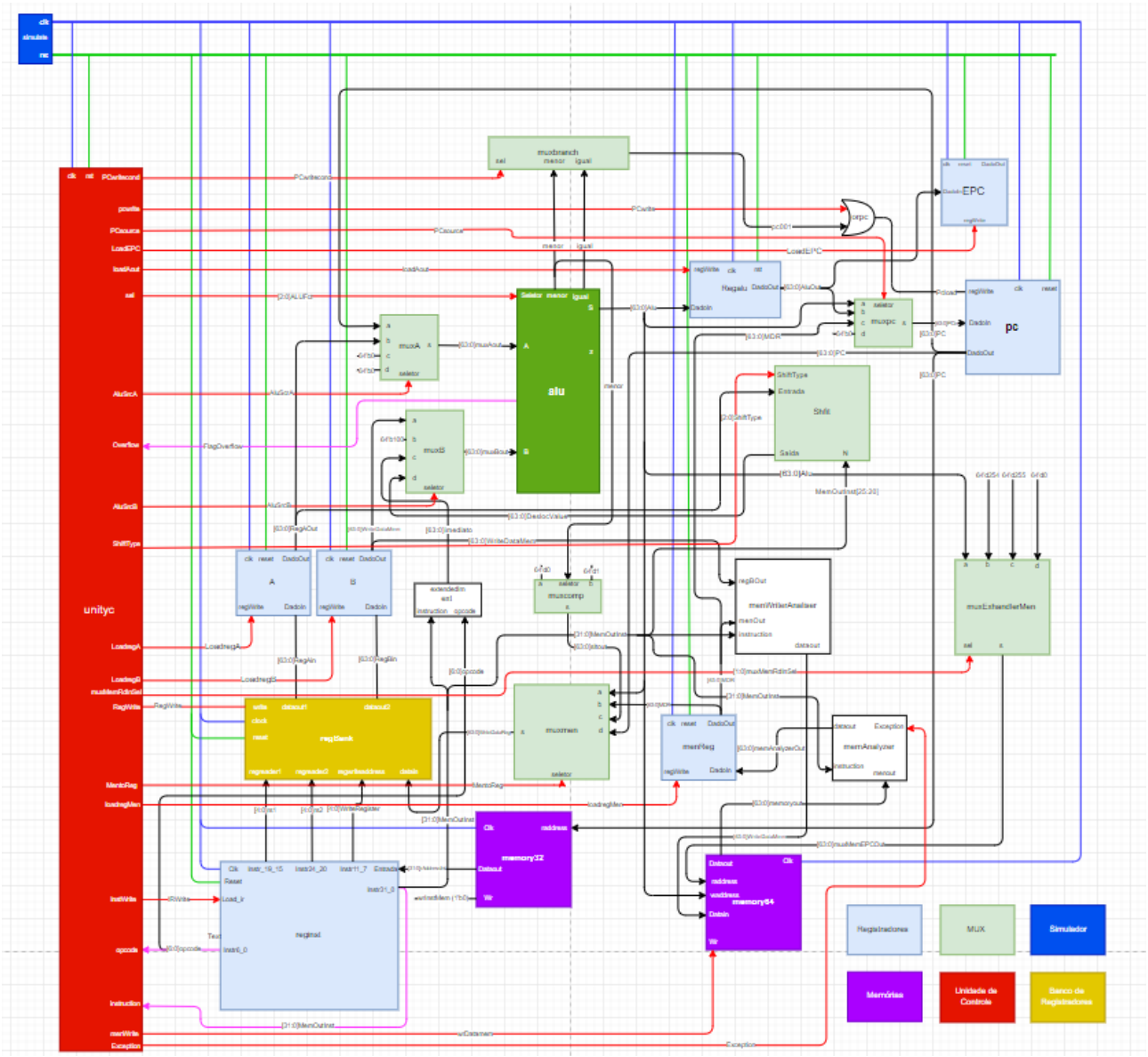
OBJETIVO:

Unir todos os módulos do sistema, é onde realmente ocorre o processamento das instruções, sendo cada etapa desse processo controlado pela Unidade de Controle.

ALGORITMO:

Conecta os módulos entre si e à Unidade de Controle. Ou seja, ele cria cabos para serem conectados entre os módulos, colocando no formato *.entrada/saída(cabo)* para cada um. Por exemplo, quando um *.saída(cabo)* existe, todos os *.entrada(cabo)* receberão o valor dessa saída, assim efetivamente conectando os módulos entre si. Caso exista mais de um *.saída(cabo)* com o mesmo "cabo", poderá haver erros, já que implica que duas saídas mudam o valor do mesmo cabo.

O conjunto dessas conexões formam o *datapath*, um diagrama que mostra visualmente, usando caixas representando os módulos e linhas representando os cabos, todas as conexões da CPU feitas na Unidade de Processamento. Ele é representado abaixo:



Um link para o arquivo do datapath, para melhor visualização, está presente no capítulo de Anexos.

1.3. Extensor de Sinal (extendSignal)

ENTRADAS:

instruction (32 bits): Entrada usada para pegar a instrução do código;

opcode (7 bits): Entrada usada para obter o *opcode*.

SAÍDAS:

extendedIm (64 bits): Imediato extendido extraído da instrução.

OBJETIVO:

Organizar e estender o imediato dado em instruction para que ele possa ser usado pelos outros módulos que requerem o tamanho 64 bits.

ALGORITMO:

Inicialmente zera a saída. Baseando-se no *opcode* e no no funct 3 (bits [14:12] de instruction) verifica o formato do imediato, o coloca na ordem certa e realiza a extensão dele, baseado no bit de sinal, e se ele é signed or unsigned.

1.4. Analisador da Saída da Memória (menAnalyser)

ENTRADAS:

instruction (32 bits): Entrada usada para pegar a instrução do código;

menout (64 bits): Entrada usada para obter a saída da memória.

Exception (1 bit): Entrada que identifica se uma exceção ocorreu nas instruções executadas.

SAÍDAS:

dataout (64 bits): Saída, com o valor da saída da memória modificado.

OBJETIVO:

Tratar a saída da memória para que os valores corretos sejam usados quando certas instruções o requerem, ou quando ocorre uma exceção no código, extendendo o sinal quando necessário.

ALGORITMO:

Inicialmente zera a saída. Basea-se no *opcode* (bits [6:0] de *instruction*) e no no *funct3* (bits [14:12] de *instruction*), e no valor de *Exception*, se *Exception* for igual a um, os oito últimos bits da saída serão os oito últimos bits da entrada *menout*, se o *opcode* e *funct3* indicarem uma instrução de load signed, pegará os últimos bits de acordo com a instrução, colocará em *dataout* e extenderá o sinal, caso seja uma instrução de load unsigned, *dataout* receberá os últimos bits de *menout* acordo com o tipo de load, mas o sinal não será extendido. Caso seja algum outro tipo de instrução, *dataout* será o próprio *menout*.

1.5. Analisador da Entrada de Escrita da Memória (menWriterAnalyser)

ENTRADAS:

instruction (32 bits): Entrada usada para pegar a instrução do código;

menout (64 bits): Entrada usada para obter a saída da memória.

RegBOut (64 bits): Entrada usada para receber a saída do registrador B.

SAÍDAS:

dataout (64 bits): Saída, com o valor para escrita na memória devidamente tratado.

OBJETIVO:

Fornecer o valor correto de escrita na memória para funções de store.

ALGORITMO:

Inicialmente iguala a saída a *menOut*. Basea-se no *opcode* (bits [6:0] de *instruction*) e no *funct3* (bits [14:12] de *instruction*). Substitui os últimos bits de *dataout* com os últimos bits de *menout* baseado em qual instrução de "store" estiver sendo executada.

2. DESCRIÇÃO DAS OPERAÇÕES

OBS1: *Extensão do Sinal

OBS2: **Completar com zeros

2.1. Tipo R

add rd, rs1, rs2

- *(rd = rs1 + rs2)*

Os valores de rs1 e rs2 são carregados nos registradores A e B, e é adicionado o valor de rs1 a rs2 na Alu. Após isso, esse resultado é então escrito no registrador rd.

sub rd, rs1, rs2

- *(rd = rs1 - rs2)*

Os valores de rs1 e rs2 são carregados nos registradores A e B, e é subtraído o valor de rs2 de rs1 na Alu. Após isso, esse resultado é então escrito no registrador rd.

and rd, rs1, rs2

- *(rd = rs1 AND rs2)*

Os valores de rs1 e rs2 são carregados nos registradores A e B, e é feito uma comparação AND bit-a-bit entre os valores de rs1 e rs2 na Alu. O resultado, então, é escrito no registrador rd.

slti rd, rs1, rs2

- *(rd = (rs1 < rs2)? 1:0)*

Os valores de rs1 e rs2 são carregados nos registradores A e B, e é feito uma comparação entre os valores de rs1 e rs2 na Alu. Caso rs1 seja menor que rs2, é colocado o valor “1” no registrador rd. Caso contrário, é colocado o valor “0”.

2.2. Tipo I

addi rd, rs1, (imm)

- *(rd = rs1 + imm *)*

O valor de rs1 é carregado no registrador A, e o imediato é decodificado e extendido para 64 bits com extensão do sinal. Então, o valor de rs1 é somado ao imediato, e o resultado é escrito no registrador rd.

slti rd, rs1, (imm)

- *(rd = (rs1 < imm) ? 1:0)*

O valor de rs1 é carregado no registrador A, e o imediato é decodificado e extendido para 64 bits com extensão do sinal. Então, o valor de rs1 é comparado ao imediato. Caso rs1 seja menor que o imediato, é colocado o valor "1" no registrador rd. Caso contrário, é colocado o valor "0".

jalr rd, rs1, (imm)

- *(rd = PC; PC = (rs1 + imm) *)*

O valor de rs1 é carregado no registrador A, e o imediato é decodificado e extendido para 64 bits com extensão do sinal. Enquanto isso ocorre, o valor atual do PC é escrito no registrador de destrino rd. Após isso, o valor de rs1 é somado ao imediato, e o resultado é escrito em PC. Mais um ciclo é usado esperando o valor de PC ser alterado apropriadamente.

lb rd, imm(rs1)

- *(rd [7:0] = MEM[rs1 + imm] *)*

O valor de rs1 é carregado no registrador A, e o imediato é decodificado e extendido para 64 bits com extensão do sinal. Ocorre a soma do valor de rs1 e o imediato, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor cortado em 1 byte, com extensão do sinal, é enviado para o registrador da memória. Finalmente, há a escrita desse valor no registrador de destino rd.

lh rd, imm(rs1)

- *(rd [15:0] = MEM[rs1 + imm] *)*

O valor de rs1 é carregado no registrador A, e o imediato é decodificado e extendido para 64 bits com extensão do sinal. Ocorre a soma do valor de rs1 e o imediato, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor cortado em 2 bytes, com extensão do sinal, é enviado para o registrador da memória. Finalmente, há a escrita desse valor no registrador de destino rd.

lw rd, imm(rs1)

- *(rd [31:0] = MEM[rs1 + imm] *)*

O valor de rs1 é carregado no registrador A, e o imediato é decodificado e extendido para 64 bits com extensão do sinal. Ocorre a soma do valor de rs1 e o imediato, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor cortado em 4 bytes, com extensão do sinal, é enviado para o registrador da memória. Finalmente, há a escrita desse valor no registrador de destino rd.

ld rd, imm(rs1)

- *(rd = MEM[rs1 + imm] *)*

O valor de rs1 é carregado no registrador A, e o imediato é decodificado e extendido para 64 bits com extensão do sinal. Ocorre a soma do valor de rs1 e o imediato, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor de 8 bytes é enviado para o registrador da memória. Finalmente, há a escrita desse valor no registrador de destino rd.

lbu rd, imm(rs1)

- *(rd [7:0] = MEM[rs1 + imm] **)*

O valor de rs1 é carregado no registrador A, e o imediato é decodificado e extendido para 64 bits com extensão do sinal. Ocorre a soma do valor de rs1 e o imediato, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor cortado em 1 byte, completado com zeros, é enviado para o registrador da memória. Finalmente, há a escrita desse valor no registrador de destino rd.

lhu rd, imm(rs1)

- *(rd [15:0] = MEM[rs1 + imm] **)*

O valor de rs1 é carregado no registrador A, e o imediato é decodificado e extendido para 64 bits com extensão do sinal. Ocorre a soma do valor de rs1 e o imediato, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor cortado em 2 bytes, completado com zeros, é enviado para o registrador da memória. Finalmente, há a escrita desse valor no registrador de destino rd.

lwu rd, imm(rs1)

- *(rd [31:0] = MEM[rs1 + imm] **)*

O valor de rs1 é carregado no registrador A, e o imediato é decodificado e extendido para 64 bits com extensão do sinal. Ocorre a soma do valor de rs1 e o imediato, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor cortado em 4 bytes, completado com zeros, é enviado para o registrador da memória. Finalmente, há a escrita desse valor no registrador de destino rd.

nop

- *No Operation*

Após a instrução ser decodificada, retorna-se imediatamente para o estado de busca de instrução, sem realizar operações.

break

- *Stop Execution*

Após a instrução ser decodificada, ela retorna para si mesma constantemente, efetivamente "quebrando" o código, impedindo de que ele continue, realizando nenhuma operação.

2.2.1. Tipo I (Shifts)

srli rd, rs1, shamt

- *rd = rs1 >> shamt (lógico)*

O valor de rs1 é carregado no registrador A, e a instrução é enviada para o módulo de extensão *ext*. Após isso, ocorre o *shift* lógico (não mantém o sinal do número) do valor de rs1 para a direita, na quantidade de shamt (de 0 a 63). O valor, após o *shift*, é então escrito no registrador rd.

srai rd, rs1, shamt

- *rd = rs1 >> shamt (aritmético)*

O valor de rs1 é carregado no registrador A, e a instrução é enviada para o módulo de extensão *ext*. Após isso, ocorre o *shift* aritmético (mantém o sinal do número) do valor de rs1 para a direita, na quantidade de shamt (de 0 a 63). O valor, após o *shift*, é então escrito no registrador rd.

slli rd, rs1, shamt

- rd = rs1 << shamt (lógico)

O valor de rs1 é carregado no registrador A, e a instrução é enviada para o módulo de extensão *ext*. Após isso, ocorre o *shift* lógico (não mantém o sinal do número) do valor de rs1 para a esquerda, na quantidade de shamt (de 0 a 63). O valor, após o *shift*, é então escrito no registrador rd.

2.3. Tipo S

sd rs2, imm(rs1)

- MEM[rs1 + imm] = rs2

São carregados os valores de rs1 e rs2 para os registradores A e B, respectivamente, e a decodificação do imediato da instrução. Ocorre a soma do imediato com o valor de rs1, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor de 8 bytes é enviado para o registrador da memória. Esse valor, junto com o valor de rs2 presente no registrador B, tem seus bytes trocados baseado na instrução dentro o *menWriterAnaliser* (No caso do sd, não há a troca). Finalmente, o valor no endereço determinado pela soma de rs1 e o imediato é sobrescrito pelo valor presente em *memWriterAnaliser*.

sw rs2, imm(rs1)

- MEM[rs1 + imm] = rs2[31:0]

São carregados os valores de rs1 e rs2 para os registradores A e B, respectivamente, e a decodificação do imediato da instrução. Ocorre a soma do imediato com o valor de rs1, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor cortado em 4 bytes é enviado para o registrador da memória. Esse valor, junto com o valor de rs2 presente no registrador B, tem seus bytes trocados baseado na instrução dentro o *menWriterAnaliser*. Finalmente, o valor no endereço determinado pela soma de rs1 e o imediato é sobrescrito pelo valor presente em *memWriterAnaliser*.

sh rs2, imm(rs1)

- MEM[rs1 + imm] = rs2[15:0]

São carregados os valores de rs1 e rs2 para os registradores A e B, respectivamente, e a decodificação do imediato da instrução. Ocorre a soma do imediato com o valor de rs1, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor cortado em 2 bytes é enviado para o registrador da memória. Esse valor, junto com o valor de rs2 presente no registrador B, tem seus bytes trocados baseado na instrução dentro o *menWriterAnaliser*. Finalmente, o valor no endereço determinado pela soma de rs1 e o imediato é sobrescrito pelo valor presente em *memWriterAnaliser*.

sb rs2, imm(rs1)

- MEM[rs1 + imm] = rs2[7:0]

São carregados os valores de rs1 e rs2 para os registradores A e B, respectivamente, e a decodificação do imediato da instrução. Ocorre a soma do imediato com o valor de rs1, e então é carregado o valor presente no endereço determinado por esse valor para o *memAnalyzer*, que determina a quantidade de bytes que serão lidos, e o valor cortado em 1 byte é enviado para o registrador da memória. Esse valor, junto com o valor de rs2 presente no registrador B, tem seus bytes trocados baseado na instrução dentro o *menWriterAnaliser*. Finalmente, o valor no endereço determinado pela soma de rs1 e o imediato é sobrescrito pelo valor presente em *memWriterAnaliser*.

2.4. Tipo SB

beq rs1, rs2, imm

*- PC = PC + (imm[12:1][0])*2 se rs1 == rs2*

São carregados os valores de rs1 e rs1 para os registradores A e B, respectivamente, para fazer a comparação, calcula-se o endereço do desvio e o carregamos no registrador da Alu. No próximo ciclo, comparamos o valores anteriormente carregados nos registradores na estrutura muxbranch e em caso positivo (os valores são iguais), escrevemos no PC.

bne rs1, rs2, imm

*- PC = PC + (imm[12:1][0])*2 se rs1 != rs2*

São carregados os valores de rs1 e rs1 para os registradores A e B, respectivamente, para fazer a comparação, calcula-se o endereço do desvio e o carregamos no registrador da Alu. No próximo ciclo, comparamos o valores anteriormente carregados nos registradores na estrutura muxbranch e em caso positivo (os valores são diferentes), escrevemos no PC.

bge rs1, rs2, imm

*- PC = PC + (imm[12:1][0])*2 se rs1 >= rs2*

São carregados os valores de rs1 e rs1 para os registradores A e B, respectivamente, para fazer a comparação, calcula-se o endereço do desvio e o carregamos no registrador da Alu. No próximo ciclo, comparamos o valores anteriormente carregados nos registradores na estrutura muxbranch e em caso positivo (o valor de rs1 é maior ou igual a rs2), escrevemos no PC.

blt rs1, rs2, imm

*- PC = PC + (imm[12:1][0])*2 se rs1 < rs2*

São carregados os valores de rs1 e rs1 para os registradores A e B, respectivamente, para fazer a comparação, calcula-se o endereço do desvio e o carregamos no registrador da Alu. No próximo ciclo, comparamos o valores anteriormente carregados nos registradores na estrutura muxbranch e em caso positivo (o valor de rs1 é menor que o de rs2), escrevemos no PC.

2.5. Tipo UJ

jal rd, imm

*rd = PC; PC = PC + (imm[20:1][0])*2*

Inicialmente, é salvo em rd o valor de PC atual e, ainda no mesmo ciclo, o valor do imediato é decodificado e extendido para 64 bits, e então o valor da soma do PC + imediato é calculado pela *Alu* e colocado em *muxpc*, e há a escrita desse valor no PC. Mais um ciclo é usado esperando o valor de PC ser alterado apropriadamente.

2.6. Tipo U

lui rd, imm

rd = {32'bimm<31>,imm,12'b0}

Após a decodificação da instrução, o sinal do imediato é estendido em mais 12 bits na unidade *ext* e imediatamente depois, o valor é enviado para *Alu* onde é somado com o valor 0. Esse valor, então, é escrito no registrador de destino rd.

3. DESCRIÇÃO DOS ESTADOS

3.1. Estados intermediários

reset

Estado inicial, define tudo como 0. Caso seja iniciado a partir do rst ser definido como 1, ele redefine tudo a 0 para reiniciar a CPU.

sumPC

Estado de busca, e onde ocorre a soma inicial do PC + 4 antes de cada função.

DecInstuction

Estado que define qual é a próxima instrução a partir do valor do opcode e dos functs, e escolhe qual será o próximo estado com isso.

3.2. Estados de Tipo R

add1

Carrega os valores de rs1 e rs2 nos registradores A e B, respectivamente.

add2

Define o MUX do registrador A e B para selecionarem os registradores em si, define a *Alu* para o modo de soma. Escreve o resultado no registrador de destino rd.

sub1

Carrega os valores de rs1 e rs2 nos registradores A e B, respectivamente.

sub2

Define o MUX do registrador A e B para selecionarem os registradores em si, define a *Alu* para o modo de subtração. Escreve o resultado no registrador de destino rd.

and1

Carrega os valores de rs1 e rs2 nos registradores A e B, respectivamente.

and2

Define a *Alu* para o modo de comparação AND, define o MUX dos registradores A e B para selecionarem os registradores em si. Escreve o resultado no registrador de destino rd.

slt1

Carrega os valores de rs1 e rs2 nos registradores A e B, respectivamente.

slt2

Define a *Alu* para o modo de comparação, coloca o MUX do registrador A para selecionar o registrador A, coloca o MUX do registrador B para selecionar o registrador B e coloca o MUX da memória para selecionar a saída do módulo de comparação. Escreve o resultado 1 ou 0 no registrador de destino rd.

3.2. Estados de Tipo I

addi1

Carrega o valor de rs1 no registrador A.

addi2

Define a *Alu* para o modo de soma, define o MUX do registrador A para selecionar o registrador A, e o MUX do registrador B para selecionar o imediato definido na instrução. Escreve o resultado no registrador de destino rd.

slti1

Carrega os valores de rs1 e rs2 nos registradores A e B, respectivamente.

slti2

Define a Alu para o modo de comparação, coloca o MUX do registrador A para selecionar o registrador A, coloca o MUX do registrador B para selecionar o imediato e coloca o MUX da memória para selecionar a saída do módulo de comparação. Escreve o resultado 1 ou 0 no registrador de destino rd.

jalr1

Carrega o valor de rs1 no registrador A. Permite escrever no registrador de destino rd. Coloca o MUX da memória para selecionar o valor de PC, e escreve esse valor em rd.

jalr2

Define a Alu para o modo de soma. Permite que o PC seja escrito, coloca o MUX do registrador A para o valor do registrador A e oloca o MUX do registrador B para o valor do imediato.

jalr3

Não faz nada. Serve para dar tempo do PC ser somado corretamente.

ld1

Carrega o valor de rs1 no registrador A.

ld2

Define a *Alu* para o modo de soma, coloca para o MUX do registrador A selecionar o valor do registrador A e coloca para o MUX do registrador B selecionar o valor do imediato.

ld3

Carrega o registrador de memória, e separa os bytes necessários. (Usado para instruções com menos de 8 bytes)

ld4

Coloca para o MUX do registrador de destino selectionar o valor do registrador da memória, e escreve o valor do registrador da memória no registrador de destino rd.

nop1

Retorna para *sumPC*, efetivamente gastando um ciclo.

break1

Retorna para si mesmo, o que deixa a CPU em um loop infinito até ser forçada a retornar para o estado *reset* quando rst = 1.

3.2.1. Estados de Tipo I (Shift)

srli1

Define a *Alu* para o modo de soma e carrega o valor de rs1 no registrador A.

srli2

Mantém a *Alu* no modo de soma, define o tipo de *shift* para o tipo da instrução, coloca para o MUX do registrador A receber o valor 0 e coloca para o MUX do registrador B receber o valor do shift. Escreve o resultado no registrador de destino rd.

srai1

Define a *Alu* para o modo de soma e carrega o valor de rs1 no registrador A.

srai2

Mantém a *Alu* no modo de soma, define o tipo de *shift* para o tipo da instrução, coloca para o MUX do registrador A receber o valor 0 e coloca para o MUX do registrador B receber o valor do shift. Escreve o resultado no registrador de destino rd.

slli1

Define a *Alu* para o modo de soma e carrega o valor de rs1 no registrador A.

slli2

Mantém a *Alu* no modo de soma, define o tipo de *shift* para o tipo da instrução, coloca para o MUX do registrador A receber o valor 0 e coloca para o MUX do registrador B receber o valor do shift. Escreve o resultado no registrador de destino rd.

3.3. Estados de Tipo S

sd1

Carrega os valores de rs1 e rs2 nos registradores A e B.

sd2

Define a *Alu* para o modo de soma, coloca para o MUX do registrador A selecionar o registrador A e coloca para o MUX do registrador B selecionar o imediato.

sd3

Mantém os valores do último estado, e carrega o registrador da memória, e separa os bytes necessários. (Usado para intrstruções com menos de 8 bytes)

sd4

Mantém os valores do último estado (com exceção do que carrega o registrador da memória) e escreve o endereço da memória definido pelo resultado da *Alu* com o valor do registrador B, baseado na instrução para escolher quantos dos 8 bytes serão escritos na memória.

3.4. Estados de Tipo SB

beq1

Define a *Alu* para o modo de soma, carrega os valores de rs1 e rs2 nos registradores A e B e carrega o resultado da *Alu* no seu registrador.

beq2

Define a *Alu* para o modo de comparação, coloca para o MUX do PC selecionar o resultado da soma, define o MUX do registrador A para selecionar o valor de PC e ativa a escrita do PC baseada na condição da instrução. Caso seja igual, o PC é escrito.

beq3

Não faz nada. Serve para dar tempo do PC ser somado corretamente.

bne1

Define a *Alu* para o modo de soma, carrega os valores de rs1 e rs2 nos registradores A e B e carrega o resultado da *Alu* no seu registrador.

bne2

Define a *Alu* para o modo de comparação, coloca para o MUX do PC selecionar o resultado da soma, define o MUX do registrador A para selecionar o valor de PC e ativa a escrita do PC baseada na condição da instrução. Caso seja diferente, o PC é escrito.

bne3

Não faz nada. Serve para dar tempo do PC ser somado corretamente.

bge1

Define a *Alu* para o modo de soma, carrega os valores de rs1 e rs2 nos registradores A e B e carrega o resultado da *Alu* no seu registrador.

bge2

Define a *Alu* para o modo de comparação, coloca para o MUX do PC selecionar o resultado da soma, define o MUX do registrador A para selecionar o valor de PC e ativa a escrita do PC baseada na condição da instrução. Caso seja maior ou igual, o PC é escrito.

bge3

Não faz nada. Serve para dar tempo do PC ser somado corretamente.

blt1

Define a *Alu* para o modo de soma, carrega os valores de rs1 e rs2 nos registradores A e B e carrega o resultado da *Alu* no seu registrador.

blt2

Define a *Alu* para o modo de comparação, coloca para o MUX do PC selecionar o resultado da soma, define o MUX do registrador A para selecionar o valor de PC e ativa a escrita do PC baseada na condição da instrução. Caso seja menor, o PC é escrito.

blt3

Não faz nada. Serve para dar tempo do PC ser somado corretamente.

3.5. Estados de Tipo UJ

jal1

Define a Alu para o modo de soma. Permite que o PC seja escrito, e permite escrever no registrador de destino rd. Coloca o MUX do registrador A para selecionar o PC, coloca o MUX do registrador B para selecionar o imediato e coloca o MUX da memória para selecionar o valor de PC.

jal2

Não faz nada. Serve para dar tempo do PC ser somado corretamente.

3.6. Estados de Tipo U

lui1

Define a *Alu* para o modo de soma, define o MUX do registrador A para selecionar o valor 0, e o MUX do registrador B para selecionar o imediato com shift 12. Escreve o resultado no registrador de destino rd.

3.7. Estados de Exceções

Execp1

Define a Alu para o modo de subtração. Coloca o MUX do registrador A para o valor de PC e coloca o MUX do registrador B para o valor 4. Escreve no registrador da *Alu*. Dependendo do tipo de exceção, define *muxMemEPCSel* como 1 ou 2.

Execp2

Mantém os valores de *sel*, *AluScrA* e *AluScrB* do último estado. Carrega o valor de EPC baseado em *muxMemEPCSel*.

Execp3

Coloca o valor de LoadEPC de volta pra 0, e mantém os valores de *sel*, *AluScrA* e *AluScrB* do último estado.

Execp4

Coloca o MUX do PC para o valor do registrador da *Alu*, e carrega no registrador da memória o valor do endereço da exceção.

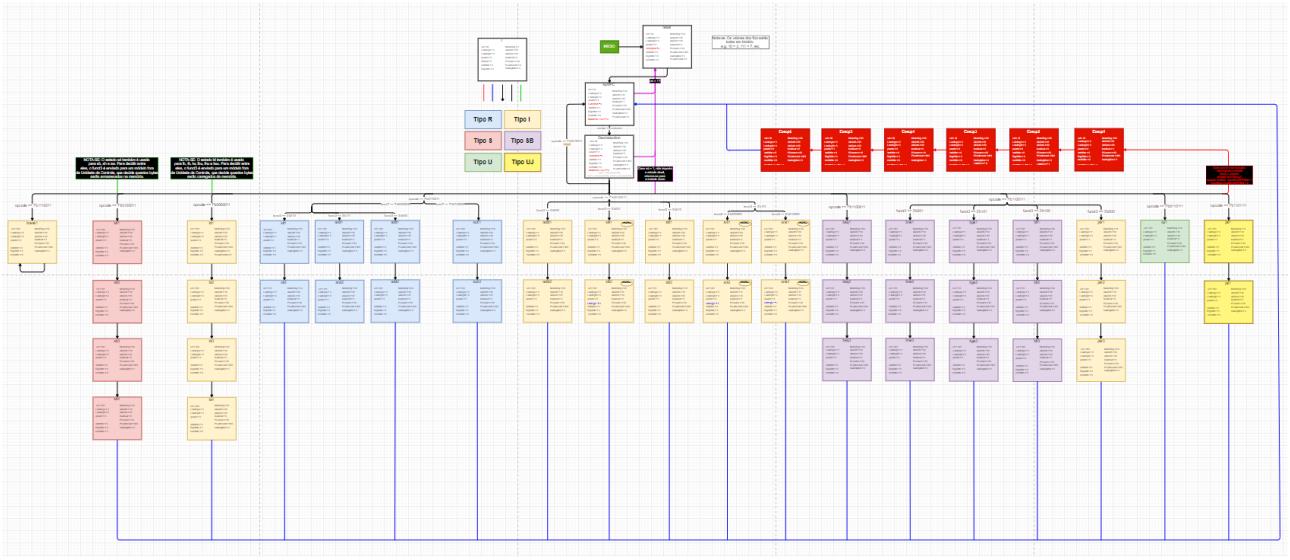
Execp5

Permite que PC seja escrito, e escreve em PC o valor do registrador da memória.

Execp6

Não faz nada. Serve para dar tempo do PC ser somado corretamente.

A máquina de estados, representando a ordem, interação e alterações de cada estado está representada abaixo:



Um link para o arquivo da máquina de estados, para melhor visualização, está presente no capítulo de Anexos.

4. ANEXOS

- Arquivos draw.io do datapath e da máquina de estados, junto com imagens de melhor resolução dos mesmos, estão presentes em: https://github.com/cingfr/CPUDiagramas