UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

RELATÓRIO DO PROJETO DE UM SUBCONJUNTO DE INSTRUÇÕES DO PROCESSADOR RISC-V

IF674 - INFRA-ESTRUTURA DE HARDWARE

Grupo 8: Amanda Soares de Castro Moraes (ascm)
Ítalo Henrique Leça da Silva (ihls2)
Maria Eduarda dos Santos Pires da Silva (mesps)
Samuel Brasileiro dos Santos Neto (sbsn)
Tales Tomaz Alves (tta)

Professora: Edna Natividade da Silva Barros (ensb)
Monitor: Lucas Eliseu de Amorim (lea)

Recife, 30 de Abril de 2019

Índice

Descrição dos Módulos

Módulo:

Sign_Extended

- **Entradas:**
  - ➜ *Instruction (32 bits):*
    Código da instrução que está sendo executada.
  - ➜ *opcode (7 bits):*
    Código da operação.

- **Saída:**
  - ➜ *immExtended (64 bits):*
    Vetor de bits que contém o valor com o sinal estendido

- **Objetivo:**
  - ➜ Módulo criado para estender o bit sinal de vetores menores que 64 bits para adequação da especificação da arquitetura do processador de 64 bits.

- **Algoritmo:**
  - ➜ De acordo com o opcode da instrução e, em alguns casos, da função, os bits menos significativos do imediato são posicionados como especificado na especificação do projeto e os demais bits são ocupados pelo bit de sinal estendido.

Módulo:

Unidade de Controle (UC)

- **Entradas:**
  - ➜ *Instruction (32 bits):*
    Código da instrução que será executada.
  - ➜ *opcode (7 bits):*
    Código da operação, presente na instrução.
  - ➜ *WriteRegister (5 bits):*
    Registrador destino onde será escrita o resultado das operações quando necessário.
  - ➜ *outB (64 bits):*
    Saída do registrador 'regB'
  - ➜ *MDR (64 bits):*
    Registrador responsável por armazenar as informações lidas da memória de dados.
  - ➜ *Overflow (1 bit):*
    Sinal, vindo da ULA, que ativa a execução da exceção de Overflow, quando um número ultrapassa 64 bits.
  - ➜ *Negativo (1 bit):*
    Sinal, vindo da ULA, que representa se o número é negativo.
  - ➜ *Igual (1 bit):*
    Sinal, vindo da ULA, que representa se o número vindo do registrador A é igual ao do registrador B ou ao do imediato.
  - ➜ *Maior (1 bit):*
    Sinal, vindo da ULA, que representa se o número vindo do registrador A é maior que o do registrador B ou que o do imediato.
  - ➜ *Menor (1 bit):*
    Sinal, vindo da ULA, que representa se o número vindo do registrador A é menor que o do registrador B ou que o do imediato.
  - ➜ *reset (1 bit):*
    Sinal, vindo da Unidade de Processamento, que avisa se é necessário zerar os valores dos registradores, máquina de estados e recomeçar a execução.

- **Saída:**
  - ➜ *loadReg1 (1 bit):*

    Sinal enviado ao módulo 'regA', que avisa se pode atualizar, e, assim, carregar o valor de 'outRR1' à saída 'outA'.
  - ➜ *loadReg2 (1 bit):*

    Sinal enviado ao módulo 'regB', que avisa se pode atualizar, e, assim, carregar o valor de 'outRR2' à saída 'outB'.
  - ➜ *loadALU (1 bit):*

    Sinal enviado ao módulo 'ALUOut', que avisa se pode atualizar, e, assim, carregar o valor de 'S' à saída 'ALUOUT'.
  - ➜ *loadCausa (1 bit):*

    Sinal enviado ao módulo 'regCAUSA', que avisa se pode atualizar, e, assim, carregar o valor de 'saidamuxcausa' à saída 'saidaCausa'.
  - ➜ *loadEPC (1 bit):*

    Sinal enviado ao módulo 'rdaddress', que avisa se pode atualizar, e, assim, carregar o valor de 'saidamuxcausa' à saída 'saidaCausa'.
  - ➜ *MenorExt (64 bits):*

    Sinal enviado ao módulo 'regCAUSA', que avisa se pode atualizar, e, assim, carregar o valor de 'saidamuxcausa' à saída 'saidaCausa'.
  - ➜ *BnoBanco (64 bits):*

    Modificação do fio 'outB', para ser uma entrada do multiplexador 'muxbanco'.
  - ➜ *MDR2 (64 bits):*

    Modificação do MDR usada para ser enviada ao banco de registradores.
  - ➜ *loadPC (1 bit):*

    Sinal enviado ao módulo 'regPC', que avisa se pode atualizar, e, assim, carregar o valor de 'saidamuxpc' na saída 'PC'.
  - ➜ *IRWrite (1 bit):*

    Sinal enviado ao registrador da memória de instruções utilizado para permitir atualização do seu valor oriundo da leitura da memória.
  - ➜ *nrst (1 bit):*

    Sinal de reset, que, se necessário, é enviado à maioria dos módulos, para os zerar.
  - ➜ *selmux2 (1 bit):*

    Seletor da saída do multiplexador 'dale2', saída a qual é armazenada em 'saidaMux1'.
  - ➜ *RegWrite (1 bit):*

Sinal enviado ao módulo 'Banco', que avisa se pode atualizar, e, assim, carregar o valor da saída do multiplexador 'muxbanco' ao registrador destino 'rd'.

➜ *wr (1 bit):*

Sinal enviado ao módulo 'datamem' (memória de dados) avisando que a escrita na memória pode ser realizada.

➜ *loadRegMemory (1 bit):*

Sinal enviado ao módulo 'regmemory' que avisa se pode atualizar, e, assim, carregar o valor da saída da memória de dados 'MemData' no registrador destino 'MDR'.

➜ *selmuxdesl (1 bit):*

Seletor da saída do multiplexador 'muxdesl', que é armazenada em 'saidamuxdesl'.

➜ *selmuxcausa (1 bit):*

Seletor da saída do multiplexador 'muxCausa', que é armazenada em 'saidamuxcausa'.

➜ *selULA (3 bits):*

Seletor da operação que será feita na ULA.

➜ *selmux4 (2 bits):*

Seletor da saída do multiplexador 'dale4', que é armazenada em 'saidaMux2'.

➜ *selmuxbanco (2 bits):*

Seletor da saída do multiplexador 'muxbanco', que é armazenada em 'datainBanco' ligado ao banco de registradores.

➜ *selmuxgera (2 bits):*

Seletor da saída do multiplexador 'geraC', que é armazenada em 'saidamuxgera'.

➜ *selmuxpc (2 bits):*

Seletor da saída do multiplexador 'muxPC', que é armazenada em 'saidamuxpc'.

➜ *estado (5 bits):*

Armazena o atual estado apenas para uma melhor visualização.

➜ *Shift (2 bits):*

Um fio que seleciona qual operação deve-se realizar no módulo 'Deslocamento'

➜ *N_shifts (6 bits):*

Determina quantos shifts devem ser realizados no módulo 'Deslocamento'.

- **Objetivo:**
  - ➜ Módulo desenvolvido para controlar a máquina de estados do projeto.

- **Algoritmo:**
  - ➜ O código da instrução é buscado na memória de instruções e sua rotina padrão é realizada. O incremento do PC é calculado e atualizado quando a execução da instrução é finalizada para que a próxima possa ser executada. Caso ocorra uma exceção (overflow ou opcode inexistente) é executada uma rotina de tratamento de exceções e a execução de instruções é interrompida.

Módulo:

Unidade de Processamento

- **Entradas:**
  - ➜ Não possui entradas.
- **Saídas:**
  - ➜ Não possui saídas.
- **Objetivo:**
  - ➜ Módulo responsável por ligar todos os componentes do processador e gerar a frequência do clock
- **Algoritmo:**
  - ➜ Inicia a execução do programa inicializando o Program Counter em zero, ativando o sinal para resetar todos os registradores e a máquina de estados.

Descrição das Operações

Instruções tipo R:

- **add rd, rs1, rs2**

  A instrução de add será decodificada, após isso, os valores nos registradores rs1 e rs2 serão carregados e selecionados para a ULA, onde a instrução de soma será realizada. Por fim, o resultado da soma será escrito no registrador destino(rd).

- **sub rd, rs1, rs2**

  A instrução de sub será decodificada, em seguida, os valores nos registradores rs1 e rs2 serão carregados e selecionados para a ULA, onde a instrução de subtração será realizada. Por fim, o resultado da operação será escrito no registrador destino(rd).

- **and rd, rs1, rs2**

  A instrução and será decodificada, logo após, os valores dos registradores rs1 e rs2 serão carregados e selecionados para a ULA, onde ocorrerá um AND bit a bit desses valores. No fim, o resultado da operação será escrito no rd.

- **slt rd, rs1, rs2**

  A instrução slt será decodificada, logo após, os valores dos registradores rs1 e rs2 serão carregados e selecionados para a ULA, onde será verificado se o valor de rs1 é menor que rs2, se for verdade, o valor escrito em rd será 1, caso contrário, será 0.

Instruções tipo I :

- **addi rd, rs1, imm**

  Após a decodificação da instrução addi, o valor armazenado no rs1 será será carregado no registrador A e o imediato será enviado para a entrada da ULA após a extensão do sinal. Enfim, a soma será realizada e o resultado será armazenado no rd.

- **slti rd, rs1, imm**

  A instrução slti é decodificada, o valor de rs1 é carregado no registrador A,o imediato é estendido e a comparação e é selecionado para ULA. Depois, é verificado se rs1 é menor que o imediato, se for verdade, o valor escrito em rd será 1, caso contrário, será 0.

- **jalr rd, rs1, imm**

  Após a decodificação da instrução jalr, o PC atual é armazenado no rd, o rs1 é carregado no registrador A e o imediato é estendido. A ULA realiza a adição dos valores do registrador A e do imediato e grava o resultado em PC.

- **lb rd, imm(rs1)**

  A instrução é decodificada, o valor de rs1 é armazenado no registrador A e o imediato tem seu sinal estendido. Em seguida, é realizada a soma (na ULA) do valor no registrador A com o imediato estendido e o resultado é gravado no registrador ALUOUT. O valor na ALUOUT representa o endereço de memória que contém o byte a ser lido (da memória de dados), estendido de acordo com o sinal para 64 bits e finalmente armazenado em rd.

- **lh rd, imm(rs1)**

  A instrução é decodificada, o valor de rs1 é armazenado no registrador A e o imediato tem seu sinal estendido. Em seguida, é realizada a soma (na ULA) do valor no registrador A com o imediato estendido e o resultado é gravado no registrador ALUOUT. O valor na ALUOUT representa o endereço de memória que contém os 16 bits (half word) a serem lidos (da memória de dados), estendidos de acordo com o sinal para 64 bits e finalmente armazenados em rd.

- **lw rd, imm(rs1)**

  A instrução é decodificada, o valor de rs1 é armazenado no registrador A e o imediato tem seu sinal estendido. Em seguida, é realizada a soma

(na ULA) do valor no registrador A com o imediato estendido e o resultado é gravado no registrador ALUOUT. O valor na ALUOUT representa o endereço de memória que contém a palavra (32 bits) a ser lida (da memória de dados), estendida de acordo com o sinal e finalmente armazenada em rd.

- **ld rd, imm(rs1)**

  A instrução é decodificada, o valor de rs1 é armazenado no registrador A e o imediato tem seu sinal estendido. Em seguida, é realizada a soma (na ULA) do valor no registrador A com o imediato estendido e o resultado é gravado no registrador ALUOUT. O valor na ALUOUT representa o endereço de memória que contém os 64 bits a serem lidos (da memória de dados) e armazenados em rd.

- **lbu rd, imm(rs1)**

  A instrução é decodificada, o valor de rs1 é armazenado no registrador A e o imediato tem seu sinal estendido. Em seguida, é realizada a soma (na ULA) do valor no registrador A com o imediato estendido e o resultado é gravado no registrador ALUOUT. O valor na ALUOUT representa o endereço de memória que contém os 8 bits a serem lidos (da memória de dados), estendidos (para 64 bits) independentemente do sinal e finalmente armazenados em rd.

- **lhu rd, imm(rs1)**

  A instrução é decodificada, o valor de rs1 é armazenado no registrador A e o imediato tem seu sinal estendido. Em seguida, é realizada a soma (na ULA) do valor no registrador A com o imediato estendido e o resultado é gravado no registrador ALUOUT. O valor na ALUOUT representa o endereço de memória que contém os 16 bits (half word) a serem lidos (da memória de dados), estendidos (para 64 bits) independentemente do sinal e finalmente armazenados em rd.

- **lwu rd, imm(rs1)**

  A instrução é decodificada, o valor de rs1 é armazenado no registrador A e o imediato tem seu sinal estendido. Em seguida, é realizada a soma (na ULA) do valor no registrador A com o imediato estendido e o resultado é gravado no registrador ALUOUT. O valor na ALUOUT representa o endereço de memória que contém a palavra (32 bits) a ser lida (da memória de dados) e estendida (para 64 bits) independentemente do sinal para finalmente poder armazenar os 64 bits em rd.

- **nop**

  A instrução é decodificada e não realiza nenhuma operação.

- **break**

  A instrução é decodificada e a execução do programa é interrompida.

Instruções tipo I (shifts):

- **srli rd, rs1, shamt**

  Após a decodificação da instrução srli, o valor de rs1 será carregado em um registrador e será feito um shift lógico à direita de shamt vezes. Enfim, esse valor será armazenado em rd.

- **srai rd, rs1, shamt**

  Após a decodificação da instrução srai, o valor de rs1 será carregado em um registrador e será feito um shift aritmético à direita de shamt vezes. Enfim, esse valor será armazenado em rd.

- **slli rd, rs1, shamt**

  Após a decodificação da instrução slli, o valor de rs1 será carregado em um registrador e será feito um shift lógico à esquerda de shamt vezes. Enfim, esse valor será armazenado em rd.

Instruções tipo S:

- **sd rs2, imm(rs1)**

  Após a decodificação da instrução sd, o valor de rs1 será carregado e selecionado para a ULA, além disso, o imediato será estendido, preservando seu sinal, e selecionado para a ULA, na qual o seu valor será somado a rs1 e salvo no registrador ALUOUT. Após essas operações, o valor de ALUOUT servirá como o endereço de memória, onde armazenaremos o valor inteiro de rs2, com 64bits (double word), que previamente foi carregado em um registrador.

- **sw rs2, imm(rs1)**

  Após a decodificação da instrução sw, o valor de rs1 será carregado e selecionado para a ULA, além disso, o imediato será estendido, preservando seu sinal, e selecionado para a ULA, na qual o seu valor será somado a rs1 e salvo no registrador ALUOUT. Após essas operações, o valor de ALUOUT servirá como o endereço de memória, onde armazenaremos a metade menos significativa do valor de rs2, com 32 bits (word), que previamente foi carregado em um registrador.

- **sh rs2, imm(rs1)**

  Após a decodificação da instrução sh, o valor de rs1 será carregado e selecionado para a ULA, além disso, o imediato será estendido, preservando seu sinal, e selecionado para a ULA, na qual o seu valor será somado a rs1 e salvo no registrador ALUOUT. Após essas operações, o valor de ALUOUT servirá como o endereço de memória, onde armazenaremos os últimos 16 bits (half word) de rs2, que previamente foi carregado em um registrador.

- **sb rs2, imm(rs1)**

  Após a decodificação da instrução sb, o valor de rs1 será carregado e selecionado para a ULA, além disso, o imediato será estendido, preservando seu sinal, e selecionado para a ULA, na qual o seu valor será somado a rs1 e salvo no registrador ALUOUT. Após essas operações, o valor de ALUOUT servirá como o endereço de memória, onde armazenaremos o byte menos significativo do valor de rs2, com 8 bits, que previamente foi carregado em um registrador.

Instruções do tipo SB:

- **beq rs1, rs2, imm**

  Primeiramente a instrução de beq será decodificada, assim como os valores dos registradores de rs1 e rs2 a partir do banco. Paralelamente o imediato terá seu sinal estendido e selecionado para ULA, da mesma maneira que PC, realizando a soma entre os dois e carregando seu valor na ALUOUT. Posteriormente, realizaremos a comparação entre rs1 e rs2 através da ULA, caso ambos sejam iguais, selecionaremos o valor de ALUOUT para o registrador PC, caso contrário PC = PC + 4.

- **bne rs1, rs2, imm**

  Primeiramente a instrução de bne será decodificada, assim como os valores dos registradores de rs1 e rs2 a partir do banco. Paralelamente o imediato terá seu sinal estendido e selecionado para ULA, da mesma maneira que PC, realizando a soma entre os dois e carregando seu valor na ALUOUT. Posteriormente, realizaremos a comparação entre rs1 e rs2 através da ULA, caso ambos sejam diferentes, selecionaremos o valor de ALUOUT para o registrador PC, caso contrário PC = PC + 4.

- **bge rs1, rs2, imm**

  Primeiramente a instrução de bge será decodificada, assim como os valores dos registradores de rs1 e rs2 a partir do banco. Paralelamente o imediato terá seu sinal estendido e selecionado para ULA, da mesma maneira que PC, realizando a soma entre os dois e carregando seu valor na ALUOUT. Posteriormente, realizaremos a comparação entre rs1 e rs2 através da ULA, caso ambos sejam iguais ou rs1 maior que rs2, selecionaremos o valor de ALUOUT para o registrador PC, caso contrário PC = PC + 4.

- **blt rs1, rs2, imm**

  Primeiramente a instrução de blt será decodificada, assim como os valores dos registradores de rs1 e rs2 a partir do banco. Paralelamente o imediato terá seu sinal estendido e selecionado para ULA, da mesma maneira que PC, realizando a soma entre os dois e carregando seu valor na ALUOUT. Posteriormente, realizaremos a comparação entre rs1 e rs2 através da ULA, caso rs1 seja menor que rs2, selecionaremos o valor de ALUOUT para o registrador PC, caso contrário PC = PC + 4.

Instruções tipo U:

- **lui rd, imm**

  De início, a instrução lui será decodificada. Após, concatenamos zero nos seus 12 bits menos significativos e estendemos seu sinal. Posteriormente, selecionamos este valor para ser armazenado em rd.

Instruções tipo UJ:

- **jal rd, imm**

  De início, a instrução jal será decodificada. Então, o valor atual de PC será armazenado em rd. Enfim, o imediato será estendido e somado ao PC.

Descrição dos Estados de Controle

- **reset_fase :**

  Reinicia todos os módulos da Unidade de Processamento que necessitam ser zerados e trocamos de estado à decode_op.

- **incrementar_pc:**

  Incrementa o PC em 4 unidades e passa para o estado atualizar_pc.

- **atualizar_pc:**

  Vai zerar o sinal para escrever em PC e desvia para ‘atualiza2’.

- **atualiza2:**

  Autorizará a escrita da saída da Memória de Instruções no registrador de instruções através do sinal ‘IRWrite’ e segue para ‘atualiza3’.

- **atualiza3:**

  IRWrite recebe o valor 0, limitando a escrita no seu respectivo registrador, após isso, iremos, finalmente, decodificar a intrução implementada.

- **decode_op:**

  Verifica, levando em consideração o opcode e os funct7 e funct 3 (quando necessário), qual o próximo estado que deve ser executado e passa o controle para ele. Caso o opcode não tenha sido identificado, significa que uma exceção foi encontrada, então uma espécie de rotina de exceção é executada.

- **tipo_R:**

  Operações do tipo R. Seleciona os registradores regA e regB como entrada da ULA, verifica a concatenação {funct7, funct3} para saber qual operação do tipo R deve ser executada mudando o seletor da ULA. Depois muda o estado para salvar_resultado.

- **tipo_I1:**

  Operações tipo I. Seleciona o regA e o imediato como entrada, seleciona a operação de adição (ULA) e vai para o estado salvar_resultado.

- **loads:**

  Seleciona o regA e o imediato como entradas da ULA, seleciona a operação de adição, carrega o valor no registrador da ALU e passa para o estado esperaDado.

- **esperaDado:**

  Estado utilizado para aguardar um ciclo para que o valor no endereço calculado anteriormente esteja pronto para ser lido e carregado no registrador da memória de dados, após essa espera, segue-se para getMemory.

- **getMemory:**

  Carrega o valor da saída da memória de dados no registrador de dados e passa para o estado MemToBanco1.

- **MemToBanco1:**

  Estado intermediário para carregar o valor corretamente no registrador de dados e passa para o estado MemToBanco2.

- **MemToBanco2:**

  Verifica, a partir do funct3, qual a instrução para carregar a posição correta da instrução no MDR2 e passa para o estado salvar_resultado.

- **tipo_S**

  Seleciona o regA e a saída do sign_extended como entradas da ALU, seleciona a operação de adição e armazena o resultado em ALUOUT. Seguido sempre do estado ‘store_intermediario’.

- **store_intermediario:**

  Carrega no registrador da Memória de Dados o valor que estava no endereço de ALUOUT e após um ciclo vai para o estado ‘store_intermediario2’.

- **store_intermediario2:**

  Identifica, a partir do campo ‘funct3’ da instrução, qual store está em execução e passa o controle para o estado correspondente.

- **store_byte:**

  Concatena-se os 56 primeiros bits da saída do registrador da Memória de Dados, calculada 2 estados anteriormente, com os 8 bits menos

significativos do regB e sinaliza através de ‘wr’ que iremos escrever na memória o resultado.

- **store_hw:**
  Concatena-se os 48 primeiros bits da saída do registrador da Memória de Dados, calculada 2 estados anteriormente, com os 16 bits menos significativos do regB e sinaliza através de ‘wr’ que iremos escrever na memória o resultado.

- **store_word:**
  Concatena-se os 32 primeiros bits da saída do registrador da Memória de Dados, calculada 2 estados anteriormente, com os 32 bits menos significativos do regB e sinaliza através de ‘wr’ que iremos escrever na memória o resultado.

- **store_double:**
  A entrada da Memória de Dados será o valor inteiro de regB, sinalizamos através de ‘wr’ que iremos escrever na memória o resultado.

- **tipo_SB:**
  Seleciona regA e regB como entradas da ALU, seleciona a operação de xor, verifica o tipo de instrução do tipo SB que deve ser executada e passa o seu respectivo estado.

- **continua_SB1:**
  Instrução de desvio beq. Verifica se os valores nos registradores são iguais e, em caso positivo, seleciona a saída da ALU como entrada do registrador PC, carrega o valor em PC e muda o estado para AntesDeInc. Em caso negativo, estado passa a ser incrementar_pc.

- **continua_SB2:**
  Instrução de desvio bne. Verifica se os valores nos registradores são diferentes e, em caso positivo, seleciona a saída da ALU como entrada do registrador PC, carrega o valor em PC e muda o estado para AntesDeInc. Em caso negativo, estado passa a ser incrementar_pc.

- **continua_SB3:**
  Instrução de desvio blt. Verifica se os valor de reg1 é menor que o reg2, em caso positivo, seleciona a saída da ALU como entrada do registrador PC, carrega o valor em PC e muda o estado para AntesDeInc. Em caso negativo, estado passa a ser incrementar_pc.

- **continua_SB4:**

  Instrução de desvio bge. Verifica se os valor de reg1 é maior ou igual que o reg2, em caso positivo, seleciona a saída da ALU como entrada do registrador PC, carrega o valor em PC e muda o estado para AntesDeInc. Em caso negativo, estado passa a ser incrementar_pc.

- **AntesDeInc:**

  Estado intermediário para esperar o valores da saída da ALU e do PC serem atualizados no momento correto.

- **shift direita:**

  Verifica se o shift deve ser aritmético ou lógico, seleciona a quantidade de shifts descrita no shamt, seleciona a saída do módulo shiftado como entrada do banco de registradores e passa o estado para salvar_resultado.

- **shift_esquerda:**

  Seleciona a quantidade de shifts descrita no shamt, seleciona a saída do módulo shiftado como entrada do banco de registradores e passa o estado para salvar_resultado.

- **slti:**

  Seleciona o regA e o imediato como entradas da ULA, seleciona a operação de menor que, carrega a saída da ALU no registrador da ALU e passa para o estado salvar_resultado.

- **jalr:**

  Seleciona o regA e o imediato como entradas da ALU, seleciona a operação de soma, seleciona a entrada do PC como a saída da ALU, o carrega e passa o estado para AntesDeInc.

- **jal:**

  Carrega ALUOUT com o resultado da soma entre o PC atual e o imediato estendido e "shiftado", seleciona o valor da saída do registrador ALUOUT para a entrada de PC e autoriza sua escrita.

- **salvar_resultado:**

  Caso não exista um overflow: verificamos qual operação deseja salvar o seu resultado em um registrador destino, dependendo da operação podemos estender o seu valor resultado ou não. Caso contrário: executamos outra espécie de rotina de exceção utilizando o estado excecoes0.

- **excecoes0:**

  Neste estado, a atualização da saída da memória de instruções (autorizada no estado ‘decode_op’) é finalizada e a escrita de seu valor no registrador de instruções é autorizada. Depois o estado passa para excecoes1.

- **excecoes1:**

  Finaliza a atualização dos valores do registrador da memória de instruções (valor no endereço 254 ou 255, a depender da exceção), do registrador EPC e do registrador causa. Em seguida, aponta para o estado excecoes2. Depois o estado passa para excecoes2.

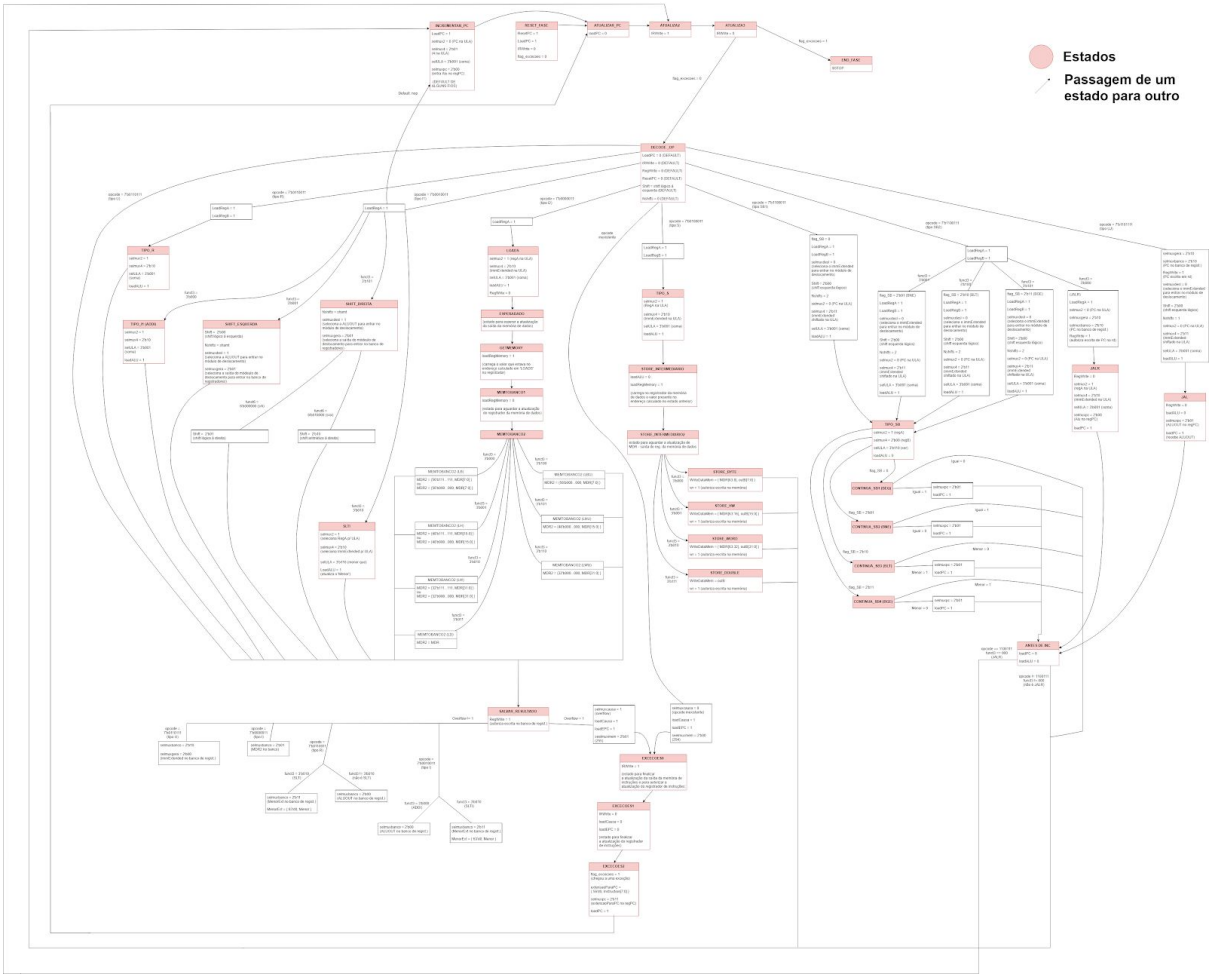
- **excecoes2:**

  Estenderá, apenas com zeros, os 8 bits menos significativos presentes no registrador da memória de instruções. No momento da atualização do PC o valor da flag_excecoes é verificado (indica a ocorrência de uma exceção) e o valor calculado anteriormente (byte estendido) é selecionado para ser o novo endereço de PC. Enfim, passa para o estado atualizar_pc. Depois o estado passa para atualizar_pc.

- **end_fase:**

  Interrompe a execução.

Máquina de Estados da Unidade de Processamento



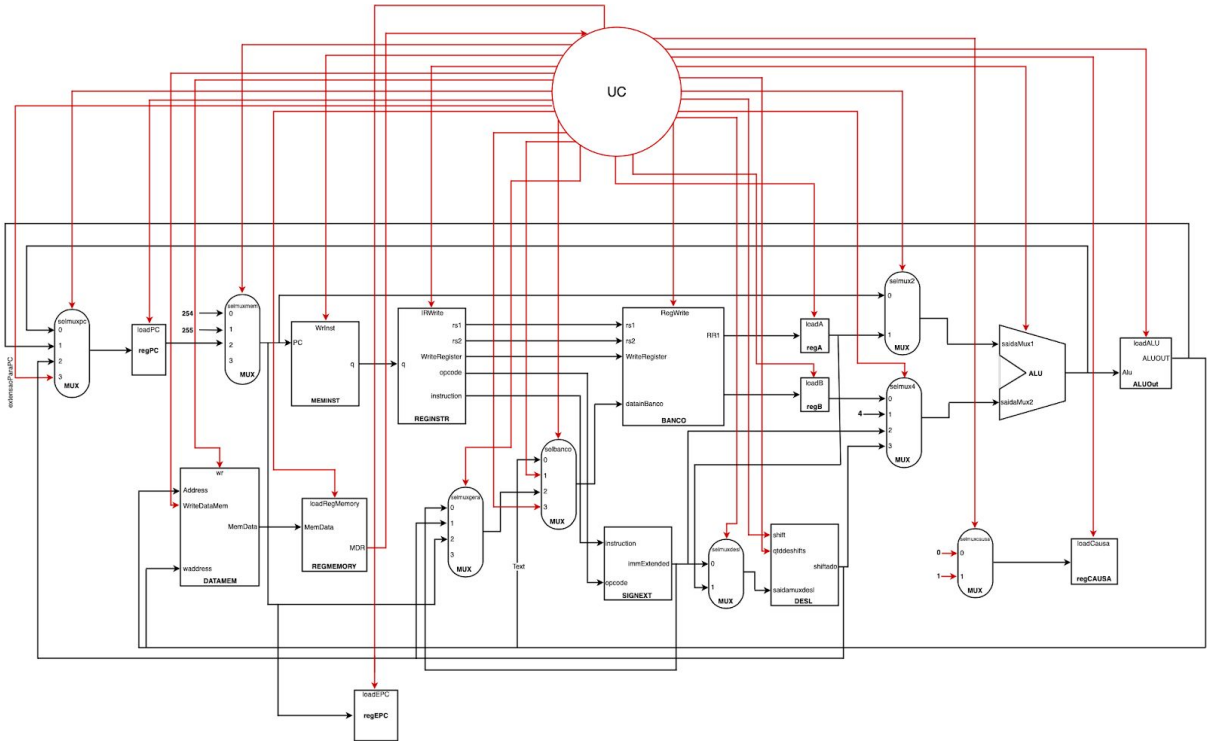
Arquivo PNG:

https://drive.google.com/open?id=1VmYKXH5lxjFAPCjCDq9XWpKh8bTXjmPe

Link Draw.io:

https://www.draw.io/?lightbox=1&highlight=FF6666&edit=_blank&layers=1&nav=1#G1GqdTSlH76CKdJ_w4Jk233Z5mFURhskrT

Diagrama da Unidade de Processamento



Arquivo PDF:

https://drive.google.com/open?id=11fZ6pGCg3H5s44VuL8u38STRI3KB0Q_y

Link Draw.io:

https://www.draw.io/?lightbox=1&highlight=FF0000&edit=_blank&layers=1&nav=1#G1snbA4Ksw3Jitgn12gyMrZlAeWXQ7H1oy