UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

RELATÓRIO DO PROJETO

ALEXANDRE DE QUEIROZ BURLE (AQB)
DANILO VAZ MARCOLINO ALVES (DVMA)
HUMBERTO LOPES FERREIRA DE SOUZA (HLFS2)
MATHEUS VINÍCIUS TEOTONIO DO NASCIMENTO ANDRADE (MVTNA)
UANDERSON RICARDO FERREIRA DA SILVA (URFS)

RECIFE
30 DE ABRIL DE 2019

SUMÁRIO

1 DESCRIÇÃO DOS MÓDULOS

1.1 ExtendSignal

Módulo: Extensor de bits do *immediate* da instrução.

Entradas:

- *in* (32 bit): Instrução atual.

Saídas:

- *out* (64 bits): Vetor de bits que estende o sinal do *immediate* de *in* para 64 bits, preenchendo os bits mais significativos em um tamanho que depende do *opcode* da instrução atual.

Objetivo:

Produzir uma saída em 64 bits do *immediate* da instrução atual.

Algoritmo:

O valor da saída, *out,* é definido de acordo com o *opcode* da instrução atual (bits[6:0] de *in*). Caso os *opcodes* do Tipo I sejam selecionados, *out* é estendida tendo os 12 bits menos significativos iguais ao *immediate* da instrução (*in[31:21]).* Caso os opcodes do Tipo S sejam selecionados, *out* é estendida tendo os 12 bits menos significativos iguais ao *immediate* da instrução (*in[31:25][11:7]).* Caso os *opcodes* do Tipo SB sejam selecionados, *out* é estendida tendo o bit menos significativo igual a ‘0’ e os próximos 12 bits iguais ao *immediate* da instrução (*in[31][7][30:25][11:8][0]).* Caso os opcodes do Tipo U sejam selecionados, *out* é estendida tendo os 12 bits menos significativos iguais a ‘0’ e os próximos 20 bits iguais ao *immediate* da instrução (*in[31:12][12’b0]).* Caso os opcodes do Tipo UJ sejam selecionados, *out* é estendida tendo o bit menos significativo igual a ‘0’ e os próximos 20 bits iguais ao *immediate* da instrução (*in[31][22:13][23][30:24][0]).*

1.2 ExtendMenor

Módulo: Extensor de bits da Saída LT da Unidade Lógica e Aritmética (ULA).

Entradas:

- *in* (1 bit): Representa o estado da flag LT da ULA.

Saídas:

- *out* (64 bits): Vetor de bits que estende o sinal de *in* para 64 bits, preenchendo os 63 bits mais significativos com zeros.

Objetivo:

Produzir uma saída em 64 bits do sinal de LT da ULA para ser escrita no banco de registradores na instrução SLT.

Algoritmo:

O valor da saída, *out,* é definida inicialmente como 64'd0 e, em seguida, é atribuído o valor de *in* ao bit menos significativo da saída (*out[0]).*

1.3 ExtendLoad

Módulo: Extensor de bits da Saída da Memória de Dados para os Loads.

Entradas:

- *Ent_mem* (64 bit): Representa os bits recebidos da Memória de Dados
- *Seletor* (3 bits): Representa qual o tipo de Load.

Saídas:

- *Saída* (64 bits): Vetor de bits que estende determinados bits de *Ent_mem* para 64 bits, de acordo com a função selecionada pelo *Seletor*.

Objetivo:

Produzir uma saída em 64 bits de um dado da Memória de Dados de acordo com o formato desejado.

Algoritmo:

O valor da saída é definido de acordo com o *Seletor.* Caso a função 0 seja selecionada, a *Saída* é igual a *Ent_mem* (*LD).* Caso a função 1 seja selecionada, a *Saída* será preenchida com os 32 bits menos significativos de *Ent_mem* e o sinal será estendido (*LW*). As funções 2 e 3 são semelhantes à função 1, porém a *Saída* será preenchida com 16 bits (*LH*) e 8 bits (*LB*) menos significativos de *Ent_mem*, respectivamente. As funções 4 (*LWU*), 5 (*LHU*) e 6 (*LBU*) são equivalentes às funções 1, 2 e 3, porém não estendem sinal, apenas preenchem a *Saída* com zeros.

1.4 ExtendStore

Módulo: Extensor de bits do registrador selecionado para os Stores.

Entradas:

- *Ent_mem* (64 bit): Representa os bits recebidos da Memória de Dados.
- *Ent_alt* (64 bit): Representa os bits recebidos do Registrador B.
- *Seletor* (2 bits): Representa qual o tipo de Store.

Saídas:

- *Saída* (64 bits): Vetor de bits que estende determinados bits do registrador B para 64 bits, de acordo com a função selecionada pelo *Seletor*, com os demais bits da *Ent_mem*.

Objetivo:

Produzir uma saída em 64 bits que seja o preenchimento de uma parte escolhida de um dado lido na Memória de Dados pelo valor correspondente do Registrador B (que é uma saída do Banco de Registradores), para ser salvo novamente na Memória de Dados.

Algoritmo:

O valor da saída é definido de acordo com o *Seletor.* Caso a função 0 seja selecionada, os 32 bits mais significativos da *Saída* serão iguais a *Ent_mem* e os outros 32 iguais a *Ent_alt* (*SW).* As funções 1 e 2 são semelhantes, porém preencherão apenas os 16 bits (*SH*) e os 8 bits (*SB*) menos significativos com a *Ent_alt*, mantendo o resto igual a *Ent_mem*, respectivamente.

2 DESCRIÇÃO DAS OPERAÇÕES

2.1 Instrução *add rd,rs1,rs2*

Após identificar a instrução *add* no estágio de decodificação, os valores dos registradores *rs1* e *rs2* são carregados a partir do banco de registradores e a operação de soma é realizada na ULA de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, o resultado é escrito no registrador de destino *rd* e uma nova instrução é buscada na memória.

2.2 Instrução *sub rd,rs1,rs2*

Após identificar a instrução *sub* no estágio de decodificação, os valores dos registradores *rs1* e *rs2* são carregados a partir do banco de registradores e a operação de subtração é realizada na ULA de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, o resultado é escrito no registrador de destino *rd* e uma nova instrução é buscada na memória.

2.3 Instrução *and rd,rs1,rs2*

Após identificar a instrução *and* no estágio de decodificação, os valores dos registradores *rs1* e *rs2* são carregados a partir do banco de registradores e a operação de AND bit a bit é realizada na ULA de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, o resultado é escrito no registrador de destino *rd* e uma nova instrução é buscada na memória.

2.4 Instrução *slt rd,rs1,rs2*

Após identificar a instrução *slt* no estágio de decodificação, os valores dos registradores *rs1* e *rs2* são carregados a partir do banco de registradores e a operação de comparação é realizada na ULA de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, o resultado é escrito no registrador de destino *rd* a partir do módulo ExtendMenor e uma nova instrução é buscada na memória.

2.5 Instrução *addi rd,rs1,imm*

Após identificar a instrução *addi* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, o valor do *imm* é carregado a partir

do módulo ExtendSignal e a operação de soma é realizada na ULA de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, o resultado é escrito no registrador de destino *rd* e uma nova instrução é buscada na memória.

2.6 Instrução *slti rd,rs1,imm*

Após identificar a instrução *slti* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, o valor do *imm* é carregado a partir do módulo ExtendSignal e a operação de comparação é realizada na ULA de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, o resultado é escrito no registrador de destino *rd* a partir do módulo ExtendMenor e uma nova instrução é buscada na memória.

2.7 Instrução *jalr rd,rs1,imm*

Após identificar a instrução *jalr* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores. A ALU carrega o PC na saída, obtendo o valor do PC no momento do *jalr*, e o escreve no registrador *rd.* O valor do *imm* é carregado a partir do módulo ExtendSignal e somado ao valor de *rs1* pela ALU. Após isso, o resultado é escrito em PC e uma nova instrução é buscada na memória.

2.8 Instrução *lb rd,imm(rs1)*

Após identificar a instrução *lb* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, um multiplexador seleciona o valor do *imm* (que foi pré-processado pelo módulo ExtendSignal) e a operação de soma é realizada na ULA entre os dois valores de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, a memória de dados lê o endereço de memória dado pelo resultado da soma, estende os 8 bits lidos (através do módulo ExtendLoad) e, finalmente, escreve os 64 bits no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.9 Instrução *lh rd,imm(rs1)*

Após identificar a instrução *lh* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, um multiplexador seleciona o valor do *imm* (que foi pré-processado pelo módulo ExtendSignal) e a operação de soma é realizada na ULA entre os dois valores de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, a Memória de Dados lê o endereço de memória

dado pelo resultado da soma, estende os 16 bits lidos (através do módulo ExtendLoad) e, finalmente, escreve os 64 bits no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.10 Instrução *lw rd,imm(rs1)*

Após identificar a instrução *lw* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, um multiplexador seleciona o valor do *imm* (que foi pré-processado pelo módulo ExtendSignal) e a operação de soma é realizada na ULA entre os dois valores de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, a Memória de Dados lê o endereço de memória dado pelo resultado da soma, estende os 32 bits lidos (através do módulo ExtendLoad) e, finalmente, escreve os 64 bits no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.11 Instrução *ld rd,imm(rs1)*

Após identificar a instrução *ld* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, um multiplexador seleciona o valor do *imm* (que foi pré-processado pelo módulo ExtendSignal) e a operação de soma é realizada na ULA entre os dois valores de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, a Memória de Dados lê o endereço de memória dado pelo resultado da soma e, finalmente, escreve os 64 bits lidos no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.12 Instrução *lbu rd,imm(rs1)*

Após identificar a instrução *lbu* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, um multiplexador seleciona o valor do *imm* (que foi pré-processado pelo módulo ExtendSignal) e a operação de soma é realizada na ULA entre os dois valores de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, a Memória de Dados lê o endereço de memória dado pelo resultado da soma, completa os 8 bits lidos com ‘0’ (através do módulo ExtendLoad) e, finalmente, escreve os 64 bits no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.13 Instrução *lhu rd,imm(rs1)*

Após identificar a instrução *lhu* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, um multiplexador seleciona o valor do *imm* (que foi pré-processado pelo módulo ExtendSignal) e a operação de soma é realizada na ULA entre os dois valores de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, a Memória de Dados lê o endereço de memória dado pelo resultado da soma, completa os 16 bits lidos com '0' (através do módulo ExtendLoad) e, finalmente, escreve os 64 bits no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.14 Instrução *lwu rd,imm(rs1)*

Após identificar a instrução *lwu* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, um multiplexador seleciona o valor do *imm* (que foi pré-processado pelo módulo ExtendSignal) e a operação de soma é realizada na ULA entre os dois valores de acordo com o sinal de controle que vem da unidade de controle. Posteriormente, a Memória de Dados lê o endereço de memória dado pelo resultado da soma, completa os 32 bits lidos com '0' (através do módulo ExtendLoad) e, finalmente, escreve os 64 bits no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.15 Instrução *nop*

Após identificar a instrução *nop* no estágio de decodificação, o sistema passa para um estado de espera. Ao término do ciclo, uma nova instrução é buscada na memória.

2.16 Instrução *break*

Após identificar a instrução *break* no estágio de decodificação, o sistema passa para um estado de loop infinito.

2.17 Instrução *srli rd,rs1,shamt*

Após identificar a instrução *srli* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, um multiplexador leva o valor em *rs1* para o módulo de Deslocamento, que tem seu resultado somado com 0 na ULA e, finalmente, escreve os 64 bits no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.18 Instrução *srai rd,rs1,shamt*

Após identificar a instrução *srai* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, um multiplexador leva o valor em *rs1* para o módulo de Deslocamento, que tem seu resultado somado com 0 na ULA e, finalmente, escreve os 64 bits no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.19 Instrução *slli rd,rs1,shamt*

Após identificar a instrução *slli* no estágio de decodificação, o valor do registrador *rs1* é carregado a partir do banco de registradores, um multiplexador leva o valor em *rs1* para o módulo de Deslocamento, que tem seu resultado somado com 0 na ULA e, finalmente, escreve os 64 bits no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.20 Instrução *sd rs2,imm(rs1)*

Após identificar a instrução *sd* no estágio de decodificação, os registradores *rs2* e *rs1* são carregados a partir do banco de registradores. O valor de *rs1* é somado ao deslocamento do *imm* (processado através do módulo ExtendSignal). Desse modo, o resultado da soma revela a posição da memória a qual iremos carregar. O valor de *rs2* é, por fim, escrito na memória. Uma nova instrução é buscada na memória.

2.21 Instrução *sw rs2,imm(rs1)*

Após identificar a instrução *sw* no estágio de decodificação, os registradores *rs2* e *rs1* são carregados a partir do banco de registradores. O valor de *rs1* é somado ao deslocamento do *imm* (processado através do módulo ExtendSignal). Desse modo, o resultado da soma revela a posição da memória a qual iremos carregar. O valor dessa posição de memória é carregado no módulo ExtendStore, que sobrescreve apenas os bits desejados (32 bits menos significativos) com *rs2*. O resultado é, por fim, escrito na memória. Uma nova instrução é buscada na memória.

2.22 Instrução *sh rs2,imm(rs1)*

Após identificar a instrução *sh* no estágio de decodificação, os registradores *rs2* e *rs1* são carregados a partir do banco de registradores. O valor de *rs1* é somado ao deslocamento do *imm* (processado através do módulo ExtendSignal). Desse modo, o

resultado da soma revela a posição da memória a qual iremos carregar. O valor dessa posição de memória é carregado no módulo ExtendStore, que sobrescreve apenas os bits desejados (16 bits menos significativos) com *rs2*. O resultado é, por fim, escrito na memória. Uma nova instrução é buscada na memória.

2.23 Instrução *sb rs2,imm(rs1)*

Após identificar a instrução *sb* no estágio de decodificação, os registradores *rs2* e *rs1* são carregados a partir do banco de registradores. O valor de *rs1* é somado ao deslocamento do *imm* (processado através do módulo ExtendSignal). Desse modo, o resultado da soma revela a posição da memória a qual iremos carregar. O valor dessa posição de memória é carregado no módulo ExtendStore, que sobrescreve apenas os bits desejados (8 bits menos significativos) com *rs2*. O resultado é, por fim, escrito na memória. Uma nova instrução é buscada na memória.

2.24 Instrução *beq rs1,rs2,imm*

Após identificar a instrução *beq* no estágio de decodificação, os valores dos registradores *rs1* e *rs2* são carregados a partir do banco de registradores. Caso *rs1* e *rs2* sejam iguais, o próximo valor do PC é calculado de acordo com o deslocamento - extensão do *imm* (processado através do módulo ExtendSignal) - e sobrescrito no PC. Caso contrário, o processo continua normalmente, sem alteração no PC. Dessa forma, uma nova instrução é buscada na memória.

2.25 Instrução *bne rs1,rs2,imm*

Após identificar a instrução *bne* no estágio de decodificação, os valores dos registradores *rs1* e *rs2* são carregados a partir do banco de registradores. Caso *rs1* e *rs2* não sejam iguais, o próximo valor do PC é calculado de acordo com o deslocamento - extensão do *imm* (processado através do módulo ExtendSignal) - e sobrescrito no PC. Caso contrário, o processo continua normalmente, sem alteração no PC. Dessa forma, uma nova instrução é buscada na memória.

2.26 Instrução *bge rs1,rs2,imm*

Após identificar a instrução *bge* no estágio de decodificação, os valores dos registradores *rs1* e *rs2* são carregados a partir do banco de registradores. Caso *rs1* seja maior ou igual a *rs2*, o próximo valor do PC é calculado de acordo com o deslocamento - extensão do *imm* (processado através do módulo ExtendSignal) - e

sobrescrito no PC. Caso contrário, o processo continua normalmente, sem alteração no PC. Dessa forma, uma nova instrução é buscada na memória.

2.27 Instrução *blt rs1,rs2,imm*

Após identificar a instrução *blt* no estágio de decodificação, os valores dos registradores *rs1* e *rs2* são carregados a partir do banco de registradores. Caso *rs1* seja menor que *rs2*, o próximo valor do PC é calculado de acordo com o deslocamento - extensão do *imm* (processado através do módulo ExtendSignal) - e sobrescrito no PC. Caso contrário, o processo continua normalmente, sem alteração no PC. Dessa forma, uma nova instrução é buscada na memória.

2.28 Instrução *lui rd,imm*

Após identificar a instrução *lui* no estágio de decodificação, o valor do *imm* é carregado a partir do módulo ExtendSignal (que põe 12 zeros à direita e estende o sinal) e o resultado é escrito no registrador de destino *rd.* Uma nova instrução é buscada na memória.

2.29 Instrução *jal rd,imm*

Após identificar a instrução *jal* no estágio de decodificação, a ALU carrega o PC na saída, obtendo o valor do PC no momento do *jal* e o escreve no registrador *rd.* O valor do *imm* é carregado a partir do módulo ExtendSignal e somado ao valor de PC pela ALU. Após isso, o resultado é escrito em PC e uma nova instrução é buscada na memória.

3 DESCRIÇÃO DOS ESTADOS DE CONTROLE

3.1 Estado *RESET*

Neste estado, todos os sinais de controle são zerados. Por definição, nenhuma operação deve ser realizada, pois é um estado de inicialização. O próximo estado é definido como *BUSCA.*

3.2 Estado *BUSCA*

Neste estado a memória de instruções busca a instrução apontada por PC e a carrega no registrador de instruções. O PC é incrementado em 4 unidades e o próximo estado é definido como *MOVTOAB*.

3.3 Estado *MOVTOAB*

Neste estado o valor dos registradores-fonte (rs1 e rs2*)* apontados pela instrução são escritos nos registradores A e B. Há também a decodificação das instruções. A partir da análise do opcode e do funct (quando necessário) é definido o próximo estado da máquina.

3.4 Estado *ADD*

Neste estado os multiplexadores que estão na entrada da ULA selecionam as saídas dos registradores A e B. A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WRBANCO.*

3.5 Estado *SUB*

Neste estado os multiplexadores que estão na entrada da ULA selecionam as saídas dos registradores A e B. A ULA realiza uma operação de subtração e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WRBANCO.*

3.6 Estado *AND*

Neste estado os multiplexadores que estão na entrada da ULA selecionam as saídas dos registradores A e B. A ULA realiza uma operação de AND bit a bit e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WRBANCO.*

3.7 Estado *SLT*

Neste estado os multiplexadores que estão na entrada da ULA selecionam as saídas dos registradores A e B. A ULA realiza uma operação de comparação e a saída menor passa pelo módulo ExtendMenor e é escrita no Banco de Registradores. O próximo estado é definido como *BUSCA.*

3.8 Estado *WRBANCO*

Neste estado o multiplexador que está na entrada de dados do Banco de Registradores seleciona a saída do registrador ALUOut. O dado então é escrito no Banco de Registradores e o próximo estado é definido como *BUSCA.*

3.9 Estado *LD*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída dos registrador A e a saída do ExtendSignal (que contém o immediate da instrução). A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WAITLD.*

3.10 Estado *WAITLD*

Neste estado o multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. Nenhuma outra operação é realizada pois é um estado que tem como finalidade aguardar a leitura do endereço pela Memória de Dados. O próximo estado é definido como *WRLDREG.*

3.11 Estado *WRLDREG*

Neste estado o seletor do ExtendLoad é definido como 0 (LD), e sua saída é escrita no Registrador da Memória de Dados. O próximo estado é definido como *WRBANCOLOAD.*

3.12 Estado *LB*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída dos registrador A e a saída do ExtendSignal (que contém o immediate da instrução). A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WAITLB*.

3.13 Estado *WAITLB*

Neste estado o multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. Nenhuma outra operação é realizada pois é um estado que tem como finalidade aguardar a leitura do endereço pela Memória de Dados. O próximo estado é definido como *WRLBREG*.

3.14 Estado *WRLBREG*

Neste estado o seletor do ExtendLoad é definido como 3 (LB), e sua saída é escrita no Registrador da Memória de Dados. O próximo estado é definido como *WRBANCOLOAD*.

3.15 Estado *LH*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída dos registrador A e a saída do ExtendSignal (que contém o immediate da instrução). A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WAITLH*.

3.16 Estado *WAITLH*

Neste estado o multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. Nenhuma outra operação é realizada pois é um estado que tem como finalidade aguardar a leitura do endereço pela Memória de Dados. O próximo estado é definido como *WRLHREG*.

3.17 Estado *WRLHREG*

Neste estado o seletor do ExtendLoad é definido como 2 (LH) e sua saída é escrita no Registrador da Memória de Dados. O próximo estado é definido como *WRBANCOLOAD.*

3.18 Estado *LW*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída dos registrador A e a saída do ExtendSignal (que contém o immediate da instrução). A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WAITLW.*

3.19 Estado *WAITLW*

Neste estado o multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. Nenhuma outra operação é realizada pois é um estado que tem como finalidade aguardar a leitura do endereço pela Memória de Dados. O próximo estado é definido como *WRLWREG.*

3.20 Estado *WRLWREG*

Neste estado o seletor do ExtendLoad é definido como 1 (LW) e sua saída é escrita no Registrador da Memória de Dados. O próximo estado é definido como *WRBANCOLOAD.*

3.21 Estado *LBU*

Neste estado, os multiplexadores que estão na entrada da ULA selecionam a saída dos registrador A e a saída do ExtendSignal (que contém o *immediate* da instrução). A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WAITLBU.*

3.22 Estado *WAITLBU*

Neste estado o multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. Nenhuma outra operação é realizada pois é um estado

que tem como finalidade aguardar a leitura do endereço pela Memória de Dados. O próximo estado é definido como *WRLBUREG.*

3.23 Estado *WRLBUREG*

Neste estado o seletor do ExtendLoad é definido como 6 (LBU) e sua saída é escrita no Registrador da Memória de Dados. O próximo estado é definido como *WRBANCOLOAD.*

3.24 Estado *LHU*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída dos registrador A e a saída do ExtendSignal (que contém o *immediate* da instrução). A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WAITLHU.*

3.25 Estado *WAITLHU*

Neste estado o multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. Nenhuma outra operação é realizada pois é um estado que tem como finalidade aguardar a leitura do endereço pela Memória de Dados. O próximo estado é definido como *WRLHUREG.*

3.26 Estado *WRLHUREG*

Neste estado o seletor do ExtendLoad é definido como 5 (LHU) e sua saída é escrita no Registrador da Memória de Dados. O próximo estado é definido como *WRBANCOLOAD.*

3.27 Estado *LWU*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída dos registrador A e a saída do ExtendSignal (que contém o *immediate* da instrução). A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WAITLWU.*

3.28 Estado *WAITLWU*

Neste estado o multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. Nenhuma outra operação é realizada pois é um estado que tem como finalidade aguardar a leitura do endereço pela Memória de Dados. O próximo estado é definido como *WRLWUREG.*

3.29 Estado *WRLWUREG*

Neste estado o seletor do ExtendLoad é definido como 4 (LWU) e sua saída é escrita no Registrador da Memória de Dados. O próximo estado é definido como *WRBANCOLOAD.*

3.30 Estado *WRBANCOLOAD*

Neste estado o multiplexador que está na entrada de dados do Banco de Registradores seleciona a saída do Registrador da Memória de Dados. O dado então é escrito no Banco de Registradores e o próximo estado é definido como *BUSCA.*

3.31 Estado *ADDI*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e do módulo ExtendSignal. A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WRBANCOADDI.*

3.32 Estado *WRBANCOADDI*

Neste estado o multiplexador que está na entrada de dados do Banco de Registradores seleciona a saída do registrador ALUOut. O dado então é escrito no Banco de Registradores e o próximo estado é definido como *BUSCA.*

3.33 Estado *SLTI*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e do módulo ExtendSignal. A ULA realiza uma operação de comparação, a saída *menor* da ULA passa pelo módulo ExtendMenor e é escrita no Banco de Registradores. O próximo estado é definido como *BUSCA.*

3.34 Estado *JALR*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador PC. A ULA carrega a entrada A na saída e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WRBANCOJALR.*

3.35 Estado WRBANCO*JALR*

Neste estado o multiplexador que está na entrada de dados do Banco de Registradores seleciona a saída do registrador ALUOut. O dado então é escrito no Banco de Registradores, os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e a saída do módulo ExtendSignal. A ULA realiza uma operação de soma e sua saída é escrita no registrador PC. O próximo estado é definido como *ENDBRANCH.*

3.36 Estado *NOP*

Neste estado, todos os sinais de controle são zerados. Nenhuma operação deve ser realizada pois é um estado que tem como finalidade não realizar nenhuma operação. O próximo estado é definido como *BUSCA.*

3.37 Estado *BREAK*

Neste estado todos os sinais de controle são zerados. Nenhuma operação deve ser realizada pois é um estado que tem como finalidade parar a execução. O próximo estado é definido como *BREAK.*

3.38 Estado *SRLI*

Neste estado o multiplexador que está na entrada do módulo de deslocamento seleciona a saída do registrador B, o seletor do módulo Shift é definido como 1 (Shift Right Lógico) e o número de deslocamentos é definido pela instrução atual. Os multiplexadores da entrada da ULA selecionam o valor 0 e a saída do módulo de deslocamento. A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WRBANCOSHIFT.*

3.39 Estado *SRAI*

Neste estado o multiplexador que está na entrada do módulo de deslocamento seleciona a saída do registrador B, o seletor do módulo Shift é definido como 2 (Shift

Right Aritmético) e o número de deslocamentos é definido pela instrução atual. Os multiplexadores da entrada da ULA selecionam o valor 0 e a saída do módulo de deslocamento. A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WRBANCOSHIFT.*

3.40 Estado *SLLI*

Neste estado o multiplexador que está na entrada do módulo de deslocamento seleciona a saída do registrador B, o seletor do módulo Shift é definido como 0 (Shift Left Lógico) e o número de deslocamentos é definido pela instrução atual. Os multiplexadores da entrada da ULA selecionam o valor 0 e a saída do módulo de deslocamento. A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WRBANCOSHIFT.*

3.41 Estado *WRBANCOSHIFT*

Neste estado o multiplexador que está na entrada de dados do Banco de Registradores seleciona a saída do registrador ALUOut. O dado então é escrito no Banco de Registradores e o próximo estado é definido como *BUSCA.*

3.42 Estado *SD*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e a saída do ExtendSignal (que contém o *immediate* da instrução). A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. O próximo estado é definido como *WRSDDATA.*

3.43 Estado WR*SDDATA*

Neste estado o multiplexador na entrada de dados da Memória de Dados seleciona a saída do registrador B, e esta é escrita na memória. O próximo estado é definido como *BUSCA*.

3.44 Estado *SW*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e a saída do ExtendSignal (que contém o *immediate* da instrução). A

ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. O próximo estado é definido como *WRSWDATA.*

3.45 Estado WR*SWDATA*

Neste estado o multiplexador na entrada de dados da Memória de Dados seleciona a saída do módulo ExtendStore, que estende os bits lidos no estado anterior e tratados pelo ExtendStore, cuja saída é escrita na memória.

3.46 Estado *SH*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e a saída do ExtendSignal (que contém o immediate da instrução). A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. O próximo estado é definido como *WRSHDATA.*

3.47 Estado WR*SHDATA*

Neste estado o multiplexador na entrada de dados da Memória de Dados seleciona a saída doExtendStore, que estende os bits lidos no estado anterior e tratados no ExtendStore, cuja saída é escrita na memória.

3.48 Estado *SB*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e a saída do ExtendSignal (que contém o immediate da instrução). A ULA realiza uma operação de soma e sua saída é escrita no registrador ALUOut. O multiplexador que está na entrada de endereço da Memória de Dados seleciona a saída de ALUOut. O próximo estado é definido como *WRSBDATA.*

3.49 Estado WR*SBDATA*

Neste estado o multiplexador na entrada de dados da Memória de Dados seleciona a saída do módulo ExtendStore, que estende os bits lidos no estado anterior e tratados pelo ExtendStore, cuja saída é escrita na memória.

3.50 Estado *BEQ*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e a saída do registrador B. A ULA realiza uma operação de comparação e, se a saída igual da ULA for 1, o próximo estado é definido como *BRANCHSUM*. Caso contrário, o próximo estado é definido como *BUSCA*.

3.51 Estado *BNE*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e a saída do registrador B. A ULA realiza uma operação de comparação e, se a saída igual da ULA for 0, o próximo estado é definido como *BRANCHSUM*. Caso contrário, o próximo estado é definido como *BUSCA*.

3.52 Estado *BLT*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e a saída do registrador B. A ULA realiza uma operação de comparação e, se a saída menor da ULA for 1, o próximo estado é definido como *BRANCHSUM*. Caso contrário, o próximo estado é definido como *BUSCA*.

3.53 Estado *BGE*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e a saída do registrador B. A ULA realiza uma operação de comparação e, se a saída menor da ULA for 0, o próximo estado é definido como *BRANCHSUM*. Caso contrário, o próximo estado é definido como *BUSCA*.

3.54 Estado *BRANCHSUM*

Neste estado, os multiplexadores que estão na entrada da ULA selecionam o valor de PC e a saída do módulo deslocamento (que contém o *immediate* extendido com um *shift left*). A ULA realiza uma operação de soma e o resultado é salvo em PC. O próximo estado é definido como *ENDBRANCH.*

3.55 Estado END*BRANCH*

Este é um estado de espera para limpar os registradores de instrução para, na próxima busca, poder carregar o endereço correto. O próximo estado é definido como *BUSCA.*

3.56 Estado *LUI*

Neste estado o multiplexador na entrada do banco de registradores seleciona a saída do módulo Extend e escreve no banco. O próximo estado é definido como *BUSCA.*

3.57 Estado *JAL*

Neste estado os multiplexadores que estão na entrada da ULA selecionam a saída do registrador PC. A ULA carrega a entrada A na saída e sua saída é escrita no registrador ALUOut. O próximo estado é definido como *WRBANCOJAL.*

3.58 Estado WR*BANCOJAL*

Neste estado o multiplexador que está na entrada de dados do Banco de Registradores seleciona a saída do registrador ALUOut. O dado então é escrito no Banco de Registradores, os multiplexadores que estão na entrada da ULA selecionam a saída do registrador A e a saída do módulo ExtendSignal. A ULA realiza uma operação de soma e sua saída é escrita no registrador PC. O próximo estado é definido como *ENDBRANCH.*

3.59 Estado *OPDEF*

Este estado é ativado ao ocorrer um opcode inexistente. Neste estado, os multiplexadores que estão na entrada da ULA selecionam a saída do registrador PC e o valor 4. A ULA realiza uma operação de subtração e sua saída é escrita no registrador EPC. O registrador de causa, por sua vez, recebe o valor 0. O mux na entrada da Memória de Dados seleciona o valor 254 e é feita a leitura. O próximo estado é definido como *WROPDEF.*

3.60 Estado *WROPDEF*

Neste estado o seletor do módulo ExtendLoad é setado em 6 para usar a função de estender com 0 os 8 bits lidos na memória, o mux na entrada do PC seleciona a saída

do ExtendLoad e esse valor é escrito em PC. O próximo estado é definido como *ENDBRANCH.*

3.61 Estado *OVER*

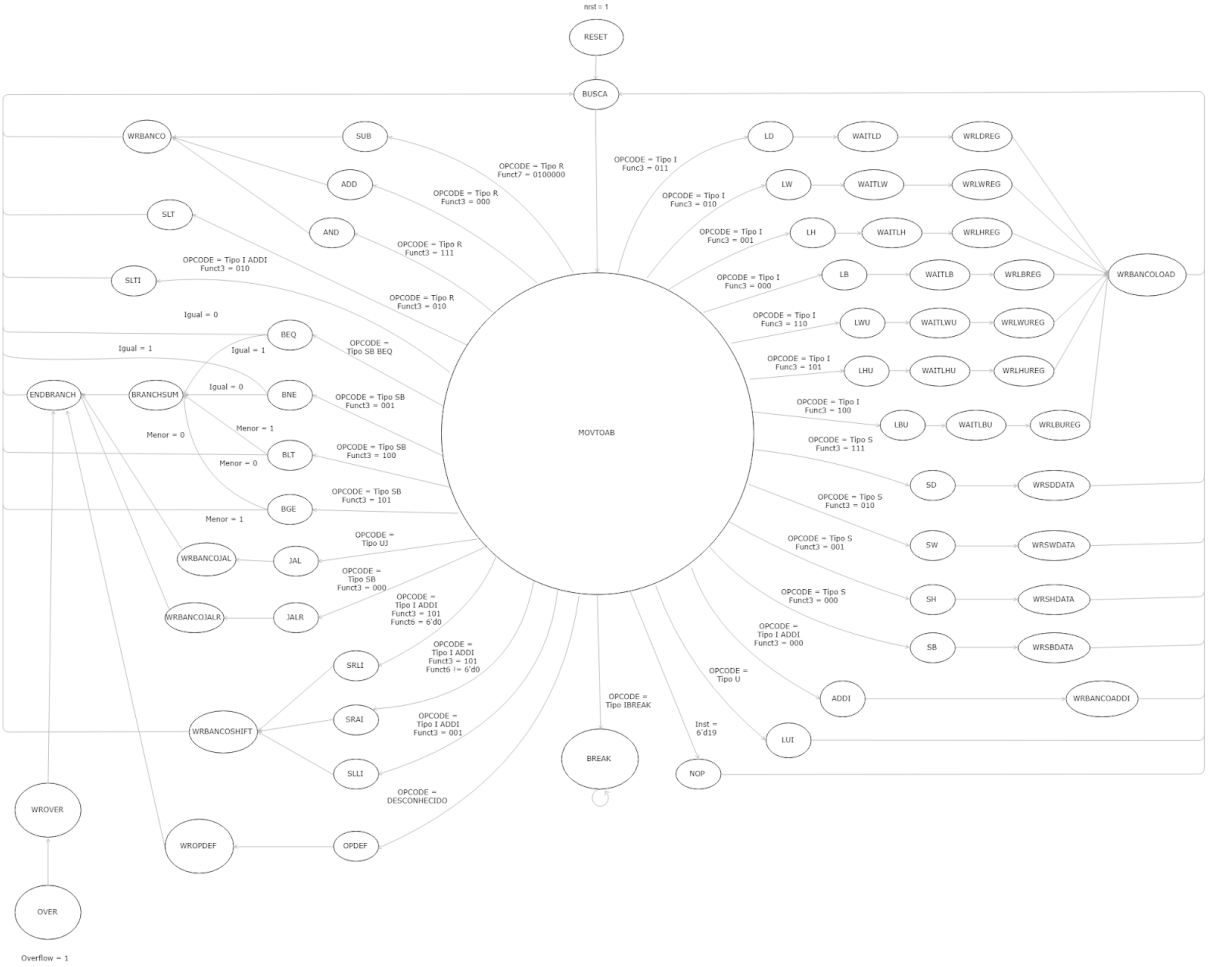
Este estado é ativado ao ocorrer um overflow. Neste estado, os multiplexadores que estão na entrada da ULA selecionam a saída do registrador PC e o valor 4. A ULA realiza uma operação de subtração e sua saída é escrita no registrador EPC. O registrador de causa, por sua vez, recebe o valor 1. O mux na entrada da Memória de Dados seleciona o valor 255 e é feita a leitura. O próximo estado é definido como *WROVER.*

3.62 Estado *WROVER*

Neste estado o seletor do módulo ExtendLoad é setado em 6 para usar a função de estender com 0 os 8 bits lidos na memória, o mux na entrada do PC seleciona a saída do ExtendLoad e esse valor é escrito em PC. O próximo estado é definido como *ENDBRANCH.*

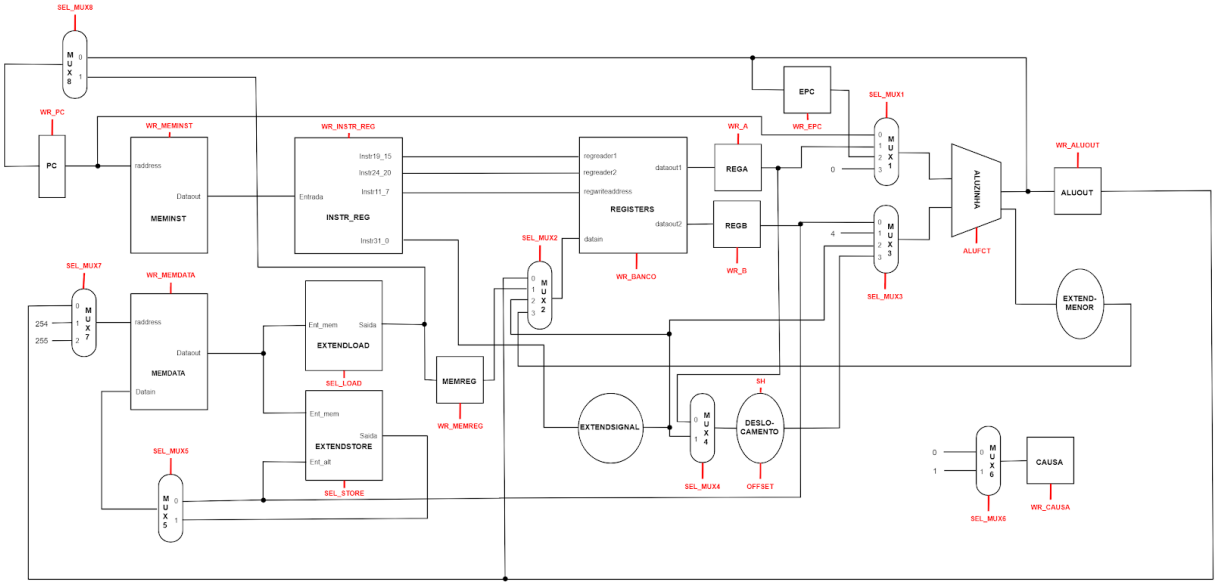
4 MÁQUINA DE ESTADOS DA UNIDADE DE PROCESSAMENTO

4.1 Diagrama da máquina de estados



Para melhor visualização:
https://drive.google.com/open?id=12SoAng5OUi6_ocj_p4CtASWnKACsSaOA

4.2 Diagrama do caminho de dados



Para melhor visualização:
https://drive.google.com/open?id=1UlT8JqCbWirP1Q-mnZHVQ0fU81gIZ2qY