Universidade Federal de Pernambuco
Centro de Informática

Relatório de Projeto

Pedro Nogueira Coutinho - pnc
Riei Joaquim Matos Rodrigues - rjmr
Victor Hugo Meirelles Silva - vhms
Victor Miguel de Morais Costa - vmmc2
Zilde Souto Maior Neto - zsmn

Recife, 30 de Abril de 2019
Professora: Edna Natividade da Silva Barros

Índice

1 Introdução

O objetivo deste relatório é demonstrar as etapas realizadas na construção do projeto da disciplina de Infraestrutura de Hardware que tem como finalidade a construção de um subconjunto de instruções do processador RISC-V.

O relatório está dividido em três partes a fim de descrever o projeto da melhor forma possível, a primeira parte consiste na descrição dos módulos usados dessa forma especificando a finalidade do uso de cada. A segunda parte é a descrição das operações onde é explicado seu comportamento e a sua implementação, já na terceira parte temos a explicação da máquina de estados.

2 Descrição das Módulos

2.1 Control Unit

Módulo : control_unit.

Entradas :

Clock: Representa o clock do sistema.

Reset: Sinal que, ao ser ativado, faz com que o processador volte para o estado inicial.

AluZero: Essa entrada indica que o resultado da operação de subtração da ALU teve como resultado o número zero. Quando isso acontece, AluZero = 1.

AluEG: Essa entrada indica se, quando aplicamos a operação de comparação na ALU entre a saída do muxA(o dado A) e a saída do muxB(o dado B) e temos como resultado que A >= B. Quando isso acontece, EG = 1.

AluLT: Essa entrada indica se, quando aplicamos a operação de comparação na ALU entre muxA(o dado A) e muxB(o dado B) temos como resultado que A < B. Quando isso acontece, temos que LT = 1.

AluO: Essa entrada indica se houve overflow ao executar uma operação na ALU. Caso isso tenha de fato ocorrido, temos que AluO = 1.

Instruction: Essa entrada recebe a instrução sendo processada atualmente, vinda do registrador de instruções (regInst).

Saidas :

PCWrite: Essa saída indica o momento no qual o valor do registrador PC pode ser atualizado. Isso acontece quando PCWrite = 1. Entretanto, quando uma instrução do tipo SB está sendo executada é possível que mesmo PCWrite seja igual a 0 seja possível atualizar o valor de PC (através da saída PCWriteCond e da saída FlagComp).

AluSrcA: Essa saída da UC vai ser responsável por selecionar qual o valor que será passado para a entrada 'A' da ALU.

LoadIr: Essa saída é responsável por permitir o carregamento da instrução retirada da memória no registrador de instruções.

LoadRegA: Essa saída é responsável por permitir o carregamento do valor vindo do banco de registradores no registrador TempRegA.

LoadRegB: Essa saída é responsável por permitir o carregamento do valor vindo do banco de registradores no registrador TempRegB.

LoadOut: Essa saída é responsável por permitir o carregamento do valor que foi calculado ao desempenhar uma operação na ALU no registrador Alu-

Out.

WriteReg: Essa saída da unidade de controle é responsável por permitir o carregamento de um certo dado no registrador especificado na instrução.

LoadRegC: Essa saída da unidade de controle é responsável por permitir o carregamento do dado proveniente da memória de dados no registrador TempRegC.

AluSrcB: Essa saída é responsável por definir qual dado será passado para a entrada 'B' da ALU.

AluFct: Essa saída é responsável por definir qual operação a ALU irá executar:

000 → Carrega o valor que sai de muxA para a saída da ALU.

001 → Realiza a soma entre o valor que sai de muxA com o que sai do muxB.

010 → Realiza a subtração entre o valor que sai de muxA com o que sai do muxB.

011 → Realiza a operação AND lógico entre o valor que sai de muxA e o valor que sai do muxB.

100 → Realiza um incremento de uma unidade no valor que sai do muxA.

101 → Realiza a negação do valor que sai do muxA.

110 → Realiza o XOR entre os valores que saem de muxA e do muxB.

111 → Realiza uma comparação entre os valores que saem de muxA e do muxB.

ResetRegA: Essa saída é responsável por permitir que o conteúdo do registrador TempRegA seja resetado, para zero, apenas quando ResetRegA = 1.

MemTOreg: Essa saída é responsável por indicar qual valor deve ser carregado no registrador de destino (Se é um valor proveniente da memória de dados, se é um valor proveniente do registrador AluOut, o valor do registrador PC ou valor vindo da saída flagRD da UC).
MemTOreg = 0 → O valor carregado no registrador de destino é proveniente do registrador Aluout.
MemTOreg = 1 → O valor carregado no registrador de destino é proveniente do registrador TempRegC.
MemTOreg = 2 → O valor carregado no registrador de destino é proveniente do registrador PC.
MemTOreg = 3 → O valor carregado no registrador de destino é proveniente da UC.

DmemControl: Essa saída é responsável por definir se é permitido escrever na memória de dados (No caso da instrução Store, DmemControl = 1) ou no caso se está travado apenas para leitura de dados (No caso da instrução Load, DmemControl = 0, assim como nas demais instruções).

ShiftControl: Essa saída é responsável por indicar qual tipo de shift será executado: shift left, shift right, shift arithmetic right ou se nenhum shift será executado. Shift Left → 00
Shift Right → 01
Shift Right Arithmetic → 10
Nenhum Shift → 11

Nshift: Essa saída vai ser responsável por indicar quantos shifts serão executados no valor de entrada.

PCWriteCond: Essa saída funciona conjuntamente com FlagComp para indicar se o novo endereço em PC deve ser o que é calculado nas instruções beq, bne, bge e blt.

FlagComp: Essa saida é utilizada para indicar se as comparações feitas nas instruções que envolvem desvios (beq, bne, bge e blt) são verdadeiras. Se forem, então FlagComp = 1. Caso contrário, FlagComp = 0.

PCSrc: Essa saída vai indicar se o PC vai receber PC + 4 direto da saída da ALU ou se vai receber o endereço calculado que está presente no registrador AluOut.

ImemWR: Essa saída é o controle de leitura e escrita do módulo de memória de instruções, que é sempre mantido em 0 (para que apenas ocorra leitura).

ShiftScr: Essa saída é responsável por definir se o dado passado para o módulo de shift será proveniente do TempRegA ou do extensor de sinal.

FlagRd: Essa saída é utilizada para escrever flags (0 ou 1) no registrador especificado na instrução(para instruções slt, slti e em caso de exceção para escrever no registrador de causa).

EndExp: Saída utilizada em caso de exceção, para buscar um endereço específico na memória de instrução, o qual contem o endereço da rotina de exceção para ser carregado em PC.
https://pt.overleaf.com/project/5cb7a4bb7b39445202595b5b exception: Essa saída é utilizada como flag para indicar ao modulo sizedadoOUT para que ele pegue apenas os 8 bits menos significativos da memData e complete o restante com zeros, para colocar na saída.

EPCcontrol: Essa saída permite o carregamento do valor vindo do módulo suber no registrador EPC.

SrcEnd: Essa saída é responsável por selecionar o endereço vindo do registrador PC ou da saída EndExp da UC, para entrada da memoria de instrução (MenInst).

srcInst: Essa saída é responsável por selecionar entre o fio PCIn (que é a saída do muxD) e o fio DMemRegC, que em caso de exceção possuirá o endereço da rotina a ser executada. A saída desse mux será carregada em PC.

CausaControl: Essa saída permite o carregamento da flag vinda da UC (0 para opcode inexistente ou 1 para overflow).

Objetivo : O objetivo desse módulo é que ele seja capaz de controlar o caminho de dados do processador por meio de sinais de controle que se comunicam com os módulos sequenciais e combinacionais de tal forma que as instruções podem ser executadas da maneira correta.

Algoritmo : O algoritmo desse módulo funciona como uma máquina de estados. Inicialmente, ela deve decodificar a instrução que foi recebida baseando-se no opcode e nos functs (funct3 e funct7) e então, irá seguir caminhos diferentes com quantidades de estados diferentes e cujas transições entre estados ocorrem de acordo com os sinais de controle provenientes da Unidade de Controle.

2.2 Extensor de Sinal

Módulo : sign_extend.

Entradas :
Instruction(32 bits) : Representa a instrução (com 32 bits) que está sendo executada no momento pelo processador.

Saída :
Extend_Instruction (64 bits) : Representa o imediato extendido depois de passar pelo processo de extensão. Nessa saída, o imediato apresenta 64 bits.

Objetivo :
O objetivo desse módulo é realizar a extensão do imediato para um novo imediato por meio da extensão de sinal, mas que agora conta com 64 bits. Entretanto, essa extensão irá ser realizada de diferentes maneiras dependendo da instrução que está sendo executada pelo processador.

Algoritmo :
Assim que o módulo recebe Instruction (32 bits) ele vai realizar a operação de extensão de formas diferentes dependendo do opcode e nos functs (funct3 e funct7) da instrução que foi passada pelo input Instruction. A checagem do opcode é feita através de um case onde verificamos os 7 bits menos significativos da instrução, já a dos functs também realizamos cases, mas a região analisada varia de acordo com a instrução. Após feito o tratamento do opcode e dos functs, nós realizamos a extensão do imediato presente na

instrução para 64 bits e esse novo imediato é disponibilizado no output. Extend_Instruction (64 bits).

2.3 Control SizeOut

Módulo : control_sizeout.

Entradas :
DadoMemOUT(64 bits) : O valor proveniente da memória de dados.
AllInst(32 bits) : A instrução proveniente do registrador de instruções.
MemData(32 bits) : Ela é a saída direta da memória de instruções, que contém o endereço da rotina a ser executada por alguma eventual exceção.
Exception(1 bit) : Flag para extender o sinal apenas com zeros em caso de exceção.

Saída :
dMemRegC (64 bits) : É a saida com o tamanho tratado para o registrador TempRegC.

Objetivo :
O objetivo desse módulo é unificar todos os loads (ld, lw, lh, lb, lbu, lhu, lwu) em uma única rotina de estado. Além de, em caso de exceção, extender corretamente o endereço de instrução para que ele seja carregado no registrador PC.

Algoritmo :
O funcionamento desse módulo é baseado no uso de um case. No caso da instrução ser um load doubleword (ld) o valor presente na memória de dados é passado diretamente para o registrador TempRegC. No caso das instruções load word (lw), load halfword (lh) e load byte (lb) os bits que não fazem parte do tamanho do dado solicitado são sobrescritos com a extensão de sinal até o 63o bit, pois o dado carregado sempre deve possuir 64 bits. No caso das instruções load word unsigned (lwu), load halfword unsigned (lhu) e load byte unsigned (lbu) ocorre o mesmo procedimento das instruções lw, lh e lb, entretanto a extensão de sinal ocorre apenas com 0.

2.4 Control SizeIn

Módulo : control_sizeIn.

Entradas :
RegBMuxB(64 bits) : É o valor presente no registrador TempRegB, que foi carregado do banco de registradores.
dMemRegC(64 bits) : É o valor presente direto da saída da memoria, no endereço previamente passado.
AllInst(32 bits) : A instrução proveniente do registrador de instruções.

Saída :
dadoMemIn (64 bits) : É o dado que deverá ser escrito na memória de dados.

Objetivo :
O objetivo desse módulo é possibilitar o store de blocos de bits menores na memória (no caso de sw, sh e sb), pois a memória é escrita apenas em blocos de 64 bits e não podemos sobrescrever regiões vizinhas do bloco de destino.

Algoritmo :
Na rotina de execução das instruções de store (sd, sw, sh, sb) foi previamente passado o endereço destino, cujo os dados presentes anteriormente nessa região ficam disponíveis em dMemRegC. Caso a instrução seja um store doubleword (sd) esse valor é ignorado e a saída do módulo para a memória é diretamente apenas o valor carregado do registrador TempRegB. No caso de instruções menores (sw, sh, sb) o valor de saída da memória de dados é lido e internamente é sobrescrito o bloco de bits específico de acordo com a instrução (sw = [31:0], sh = [15:0], sb = [7:0]). E por último esse bloco alterado de 64bits é enviado para a memória de dados.

2.5 Suber

Entradas :
PCOut(64 bits) : É a saída do registrador PC com o endereço da instrução a ser buscada da memória.

Saída :
suberOut (64 bits) : É a saída para o registrador EPC com o valor vindo

do registrador PC - 4.

Objetivo :

Esse módulo é necessário pois ao chegar no estado de decodificação para identificar opcode inexistente e a partir do qual as operações com a ALU podem ocasionar overflow, o valorde PC já foi incrementado para apontar para a instrução seguinte. Assim, ao ser identificada uma exceção, como é preciso salvar no registrador EPC o endereço da instrução que apresentou uma exceção colocamos na saída o valor PC - 4 para ser salvo em EPC.

Algoritmo :

O algoritmo desse módulo se baseia em puramente uma lógica combinacional, onde nele, a saída é igual ao valor da entrada menos 4.

3 Descrição das Operações

Nessa seção serão descritas as operações construídas no decorrer do projeto, divididas de acordo com o tipo.

3.1 Instruções do tipo R

As instruções do tipo R seguem o seguinte formato:

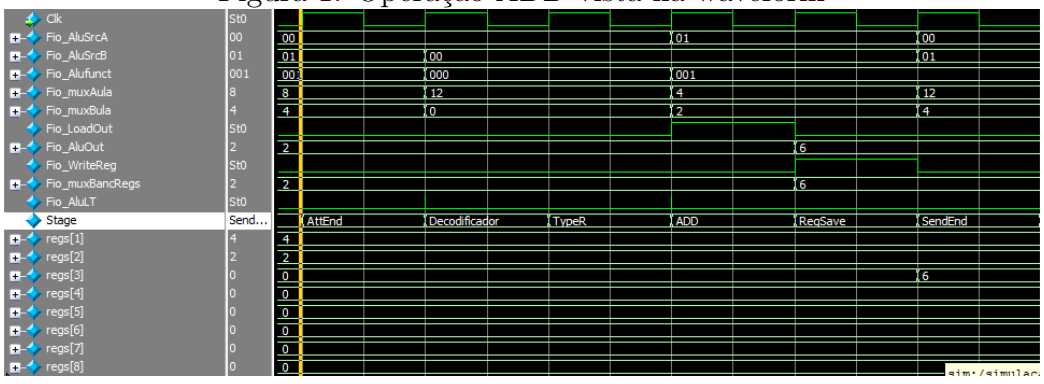
7	5	5	3	5	7
funct7	rs2	rs1	funct3	rd	opcode

3.1.1 add

add rd,rs1,rs2 $rd \leftarrow rs1 + rs2$

Após identificar a instrução add no estágio de decodificação os valores dos registradores rs1 e rs2 são carregados a partir do banco de registradores, escritos posteriormente nos registradores TempRegA e TempRegB. A operação de soma é realizada na ALU de acordo com o sinal alufct que vem da nossa unidade de controle, posteriormente o resultado é gravado no registrador de destino rd e uma nova instrução é buscada na memória no próximo ciclo.

Figura 1: Operação ADD vista na waveform



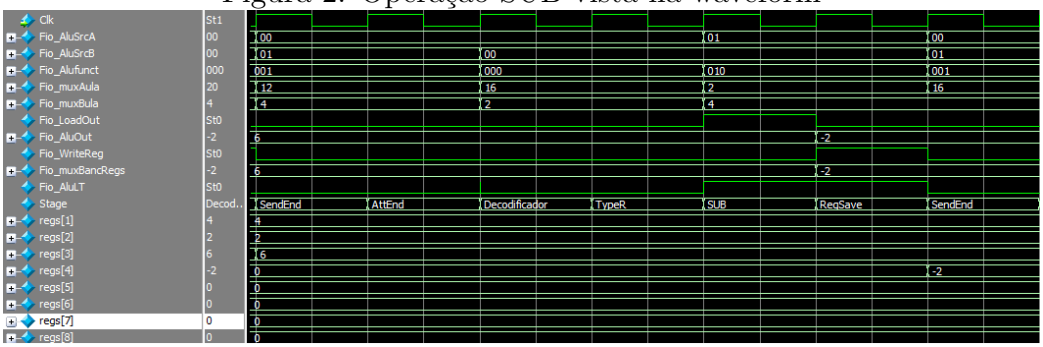
3.1.2 sub

sub rd,rs1,rs2 $rd \leftarrow rs1 - rs2$

Após identificar a instrução sub no estágio de decodificação os valores

dos registradores rs1 e rs2 são carregados a partir do banco de registradores, escritos posteriormente nos registradores TempRegA e TempRegB. A operação de subtração é realizada na ALU de acordo com o sinal alufct que vem da nossa unidade de controle, posteriormente o resultado é gravado no registrador de destino rd e uma nova instrução é buscada na memória no próximo ciclo.

Figura 2: Operação SUB vista na waveform

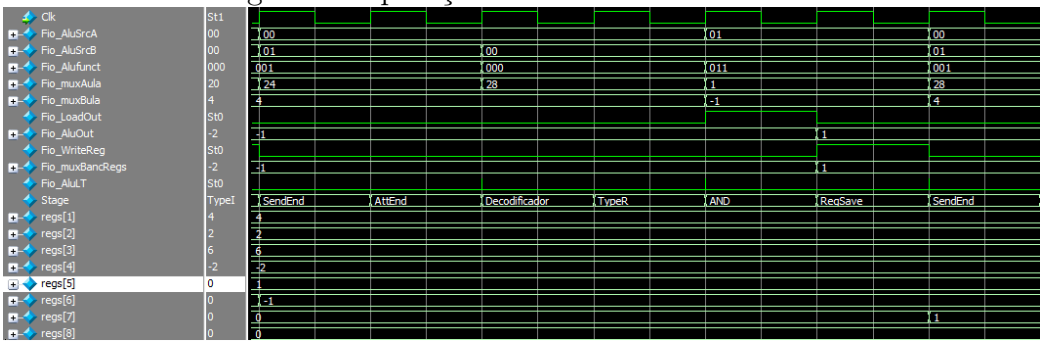


3.1.3 and

and rd,rs1,rs2 $\qquad rd \leftarrow rs1\ AND\ rs2$

Após identificar a instrução sub no estágio de decodificação os valores dos registradores rs1 e rs2 são carregados a partir do banco de registradores, escritos posteriormente nos registradores TempRegA e TempRegB. A operação de AND é realizada na ALU de acordo com o sinal alufct que vem da nossa unidade de controle, posteriormente o resultado é gravado no registrador de destino rd e uma nova instrução é buscada na memória no próximo ciclo.

Figura 3: Operação AND vista na waveform

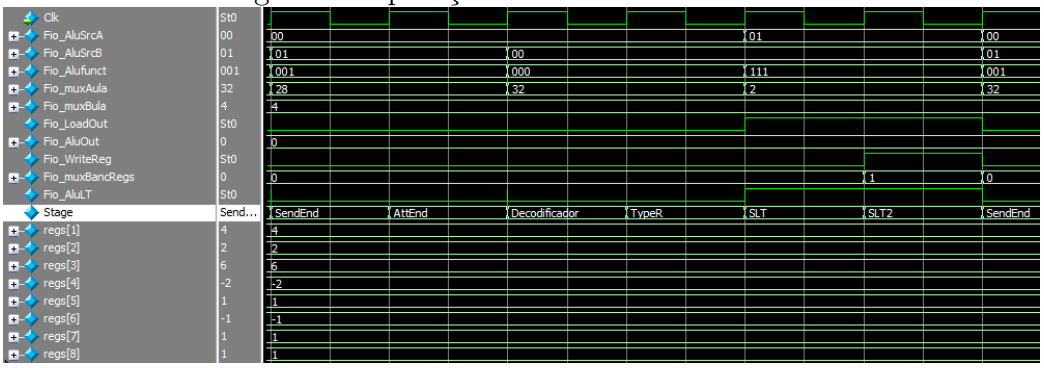


3.1.4 slt

slt rd,rs1,rs2 $rd \leftarrow (rs1 < rs2)\ ?\ 1:0$

Após identificar a instrução sub no estágio de decodificação os valores dos registradores rs1 e rs2 são carregados a partir do banco de registradores, escritos posteriormente nos registradores TempRegA e TempRegB. A operação de comparação de menor que (<) é realizada na ALU de acordo com o sinal alufct que vem da nossa unidade de controle, posteriormente o valor 1 ou 0 é gravado no registrador de destino rd, dependendo do resultado de aluOut e uma nova instrução é buscada na memória no próximo ciclo.

Figura 4: Operação SLT vista na waveform



3.2 Instruções do tipo I

As instruções do tipo I seguem o seguinte formato:

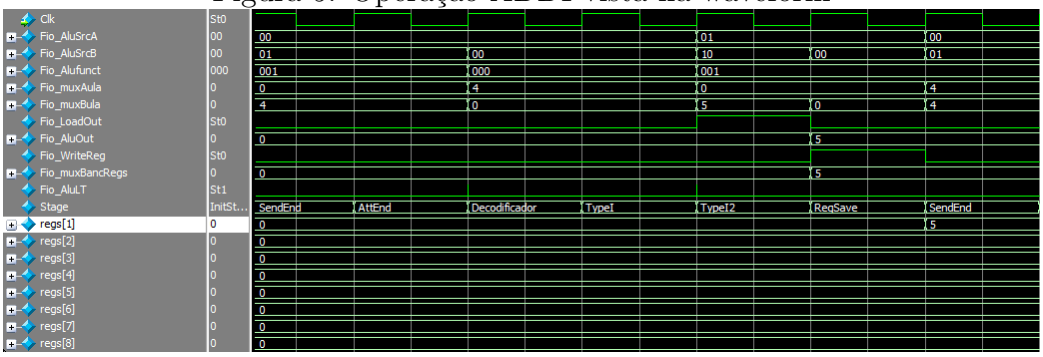
12	5	3	5	7
immediate	rs1	funct3	rd	opcode

3.2.1 addi

addi rd,rs1,imm $rd \leftarrow rs1 + imm$

Após identificar a instrução addi no estágio de decodificação o valor do registrador rs1 é carregado a partir do banco de registradores em TempRegA. A operação de soma é realizada na ALU de acordo com o sinal alufct que vem da nossa unidade de controle. O nosso imm passa pelo pelo extensor de sinal (para extendê-lo para 64bits), passamos esse resultado através do nosso muxB diretamente para a ALU, dessa forma realizando a soma do nosso rs1 com o imm. Posteriormente o resultado da operação é gravado no registrador de destino rd e uma nova instrução é buscada na memória no próximo ciclo.

Figura 5: Operação ADDI vista na waveform



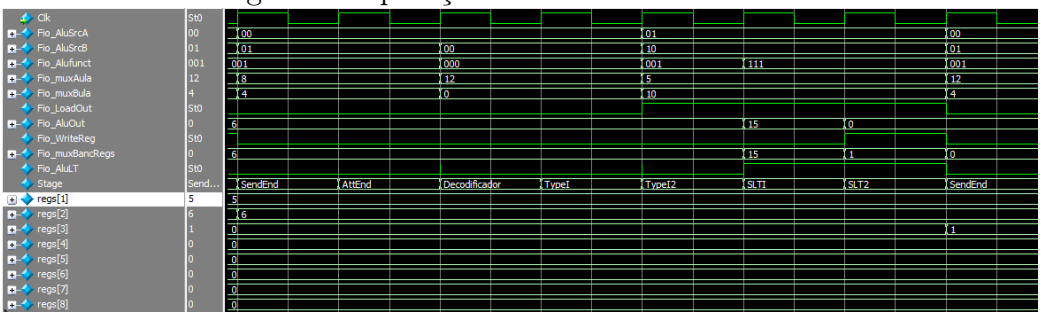
3.2.2 slti

slti rd,rs1,imm $rd \leftarrow (rs1 < imm)\ ?\ 1:0$

Após identificar a instrução slti no estágio de decodificação o valor do registrador rs1 é carregado a partir do banco de registradores em TempRegA. A operação de comparação de menor que (<) é realizada na ALU de acordo com o sinal alufct que vem da nossa unidade de controle. O nosso imm passa pelo pelo extensor de sinal (para extendê-lo para 64bits), passamos

esse resultado através do nosso muxB diretamente para a ALU, dessa forma realizando a soma do nosso rs1 com o imm. Posteriormente o o valor 1 ou 0 é gravado no registrador de destino rd, dependendo do resultado de aluOut é e uma nova instrução é buscada na memória no próximo ciclo.

Figura 6: Operação SLTI vista na waveform

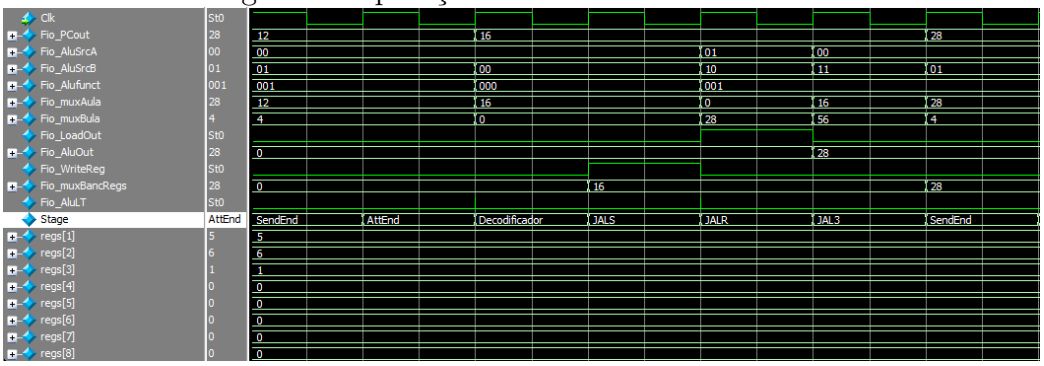


3.2.3 jalr

jalr rd,rs1,imm $rd \leftarrow PC;$ $PC \leftarrow rs1 + imm$

Após identificar a instrução jalr no estágio de decodificação, carregamos o valor de PC (já deslocado + 4) no registrador rd. O valor do registrador rs1 é carregado a partir do banco de registradores em TempRegA, simultaneamente o nosso imm passa para o extensor de sinal e posteriormente passa pelo muxB para ir até a entrada da ALU. Colocamos a ALU para a operação de soma, somamos o valor de TempRegA (rs1) com muxBula (imm) e obteremos o nosso endereço já deslocado, que será posteriormente carregado no PC.

Figura 7: Operação JALR vista na waveform

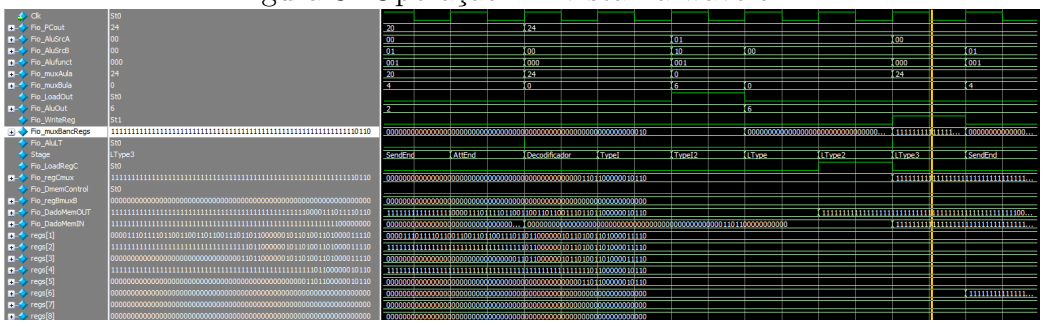


3.2.4 lb

lb rd, imm(rs1) $\qquad$ *rd[7:0]* $\leftarrow$ *MEM[rs1 + imm]*

Após identificar a instrução lb no estágio de decodificação, o valor do registrador rs1 é carregado a partir do banco de registradores e salvo em TempRegA e será passado para ALU através do muxA, de mesmo modo, o nosso imm passa pelo extensor de sinal para que logo após todo esse processo, passe pelo muxB e vá para a entrada da ALU. Através do sinal alufct da nossa unidade de controle, colocamos a operação de soma na nossa ALU e somamos rs1 que está na saída do muxA, com o imm, vindo de muxB. Dessa forma obtemos o endereço deslocado de nosso rs1, que estará em AluOut. Posteriormente a isso, vamos consultar na memória o valor que está no endereço de rs1 + imediato, pegando o fio de AluOut (portanto já somado com o deslocamento dado no imm), mas como estamos carregando um tipo byte (8bits), passamos por um módulo que vai retirar apenas a parte necessária da nossa memória, extendemos o sinal e salvamos o valor obtido da consulta no TempRegC, passando ele através do MuxC para ser salvo no registrador de destino rd, em nosso banco de registradores.

Figura 8: Operação LB vista na waveform



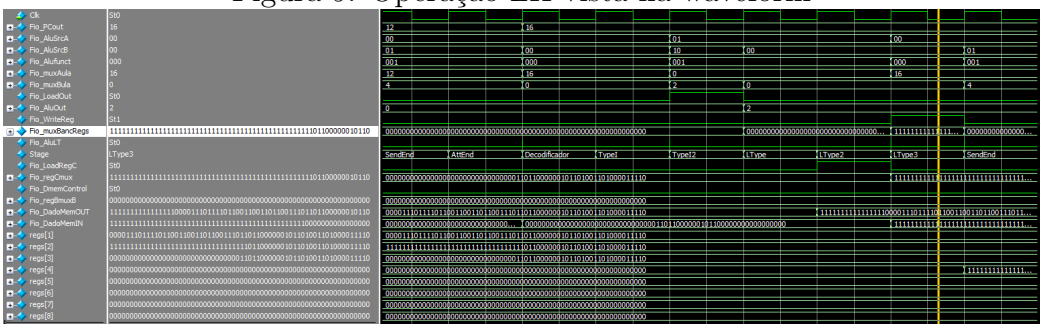
3.2.5 lh

lh rd, imm(rs1) $\qquad$ *rd[15:0]* $\leftarrow$ *MEM[rs1 + imm]*

Após identificar a instrução lh no estágio de decodificação, o valor do registrador rs1 é carregado a partir do banco de registradores e salvo em TempRegA e será passado para ALU através do muxA, de mesmo modo, o nosso imm passa pelo extensor de sinal para que logo após todo esse processo, passe pelo muxB e vá para a entrada da ALU. Através do sinal alufct da nossa unidade de controle, colocamos a operação de soma na nossa ALU e

somamos rs1 que está na saída do muxA, com o imm, vindo de muxB. Dessa forma obtemos o endereço deslocado de nosso rs1, que estará em AluOut. Posteriormente a isso, vamos consultar na memória o valor que está no endereço de rs1 + imediato, pegando o fio de AluOut (portanto já somado com o deslocamento dado no imm), mas como estamos carregando um tipo byte (16bits), passamos por um módulo que vai retirar apenas a parte necessária da nossa memória, extendemos o sinal e salvamos o valor obtido da consulta no TempRegC, passando ele através do MuxC para ser salvo no registrador de destino rd, em nosso banco de registradores.

Figura 9: Operação LH vista na waveform

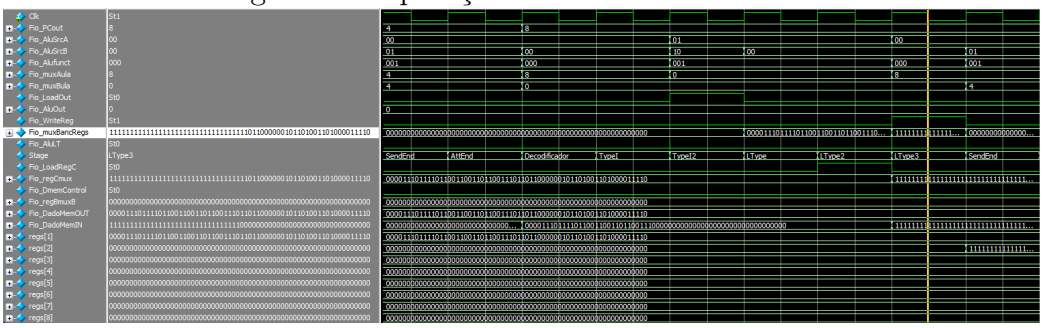


3.2.6 lw

lw rd, imm(rs1) $rd[31:0] \leftarrow MEM[rs1 + imm]$

Após identificar a instrução lw no estágio de decodificação, o valor do registrador rs1 é carregado a partir do banco de registradores e salvo em TempRegA e será passado para ALU através do muxA, de mesmo modo, o nosso imm passa pelo extensor de sinal para que logo após todo esse processo, passe pelo muxB e vá para a entrada da ALU. Através do sinal alufct da nossa unidade de controle, colocamos a operação de soma na nossa ALU e somamos rs1 que está na saída do muxA, com o imm, vindo de muxB. Dessa forma obtemos o endereço deslocado de nosso rs1, que estará em AluOut. Posteriormente a isso, vamos consultar na memória o valor que está no endereço de rs1 + imediato, pegando o fio de AluOut (portanto já somado com o deslocamento dado no imm), mas como estamos carregando um tipo byte (32bits), passamos por um módulo que vai retirar apenas a parte necessária da nossa memória, extendemos o sinal e salvamos o valor obtido da consulta no TempRegC, passando ele através do MuxC para ser salvo no registrador de destino rd, em nosso banco de registradores.

Figura 10: Operação LW vista na waveform

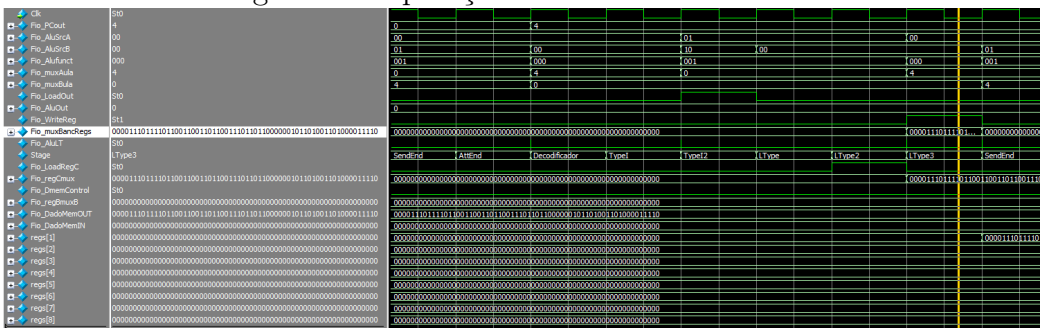


3.2.7 ld

ld rd, imm(rs1) $\qquad$ *rd* $\leftarrow$ *MEM[rs1 + imm]*

Após identificar a instrução ld no estágio de decodificação, o valor do registrador rs1 é carregado a partir do banco de registradores e salvo em TempRegA e será passado para ALU através do muxA, de mesmo modo, o nosso imm passa pelo extensor de sinal para que logo após todo esse processo, passe pelo muxB e vá para a entrada da ALU. Através do sinal alufct da nossa unidade de controle, colocamos a operação de soma na nossa ALU e somamos rs1 que está na saída do muxA, com o imm, vindo de muxB. Dessa forma obtemos o endereço deslocado de nosso rs1, que estará em AluOut. Posteriormente a isso, vamos consultar na memória o valor que está no endereço de rs1 + imediato, pegando o fio de AluOut (portanto já somado com o deslocamento dado no imm), mas como estamos carregando um tipo byte (64bits), passamos por um módulo que vai retirar apenas a parte necessária da nossa memória, nesse caso todo o bloco lido da memória, extendemos o sinal e salvamos o valor obtido da consulta no TempRegC, passando ele através do MuxC para ser salvo no registrador de destino rd, em nosso banco de registradores.

Figura 11: Operação LD vista na waveform

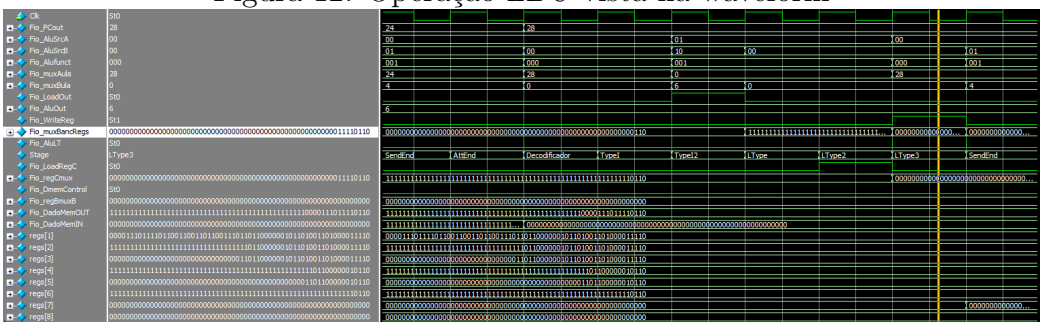


3.2.8 lbu

lbu rd, imm(rs1) $rd[7:0] \leftarrow MEM[rs1 + imm]$

Após identificar a instrução lbu no estágio de decodificação, o valor do registrador rs1 é carregado a partir do banco de registradores e salvo em TempRegA e será passado para ALU através do muxA, de mesmo modo, o nosso imm passa pelo extensor de sinal para que logo após todo esse processo, passe pelo muxB e vá para a entrada da ALU. Através do sinal alufct da nossa unidade de controle, colocamos a operação de soma na nossa ALU e somamos rs1 que está na saída do muxA, com o imm, vindo de muxB. Dessa forma obtemos o endereço deslocado de nosso rs1, que estará em AluOut. Posteriormente a isso, vamos consultar na memória o valor que está no endereço de rs1 + imediato, pegando o fio de AluOut (portanto já somado com o deslocamento dado no imm), mas como estamos carregando um tipo byte (8bits), passamos por um módulo que vai retirar apenas a parte necessária da nossa memória, estendemos o sinal apenas com zeros, e salvamos o valor obtido da consulta no TempRegC, passando ele através do MuxC para ser salvo no registrador de destino rd, em nosso banco de registradores.

Figura 12: Operação LBU vista na waveform

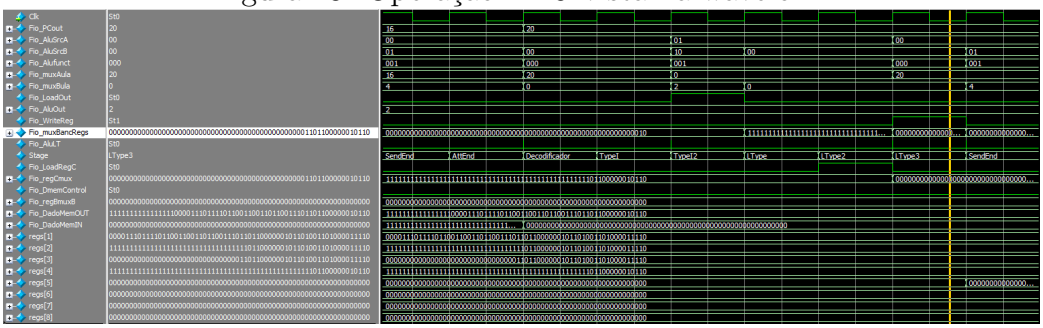


3.2.9 lhu

lhu rd, imm(rs1) $rd[15:0] \leftarrow MEM[rs1 + imm]$

Após identificar a instrução lhu no estágio de decodificação, o valor do registrador rs1 é carregado a partir do banco de registradores e salvo em TempRegA e será passado para ALU através do muxA, de mesmo modo, o nosso imm passa pelo extensor de sinal para que logo após todo esse processo, passe pelo muxB e vá para a entrada da ALU. Através do sinal alufct da nossa unidade de controle, colocamos a operação de soma na nossa ALU e somamos rs1 que está na saída do muxA, com o imm, vindo de muxB. Dessa forma obtemos o endereço deslocado de nosso rs1, que estará em AluOut. Posteriormente a isso, vamos consultar na memória o valor que está no endereço de rs1 + imediato, pegando o fio de AluOut (portanto já somado com o deslocamento dado no imm), mas como estamos carregando um tipo byte (16bits), passamos por um módulo que vai retirar apenas a parte necessária da nossa memória, estendemos o sinal apenas com zeros, e salvamos o valor obtido da consulta no TempRegC, passando ele através do MuxC para ser salvo no registrador de destino rd, em nosso banco de registradores.

Figura 13: Operação LHU vista na waveform



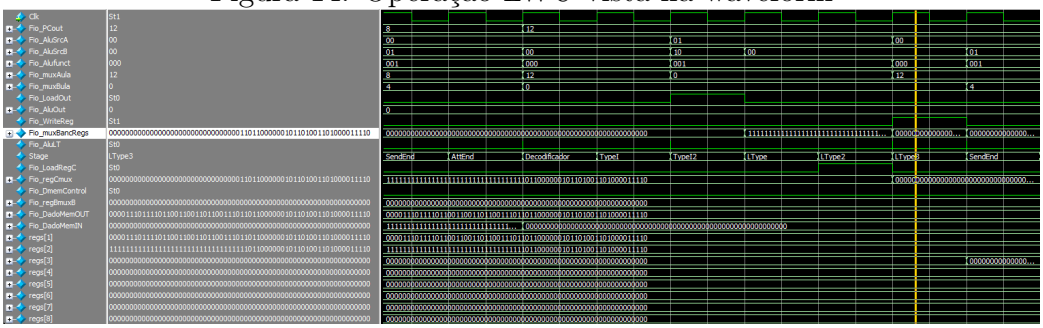
3.2.10 lwu

lwu rd, imm(rs1) $rd[31:0] \leftarrow MEM[rs1 + imm]$

Após identificar a instrução lwu no estágio de decodificação, o valor do registrador rs1 é carregado a partir do banco de registradores e salvo em TempRegA e será passado para ALU através do muxA, de mesmo modo, o nosso imm passa pelo extensor de sinal para que logo após todo esse processo, passe pelo muxB e vá para a entrada da ALU. Através do sinal alufct da nossa unidade de controle, colocamos a operação de soma na nossa ALU e

somamos rs1 que está na saída do muxA, com o imm, vindo de muxB. Dessa forma obtemos o endereço deslocado de nosso rs1, que estará em AluOut. Posteriormente a isso, vamos consultar na memória o valor que está no endereço de rs1 + imediato, pegando o fio de AluOut (portanto já somado com o deslocamento dado no imm), mas como estamos carregando um tipo byte (32bits), passamos por um módulo que vai retirar apenas a parte necessária da nossa memória, estendemos o sinal apenas com zeros, e salvamos o valor obtido da consulta no TempRegC, passando ele através do MuxC para ser salvo no registrador de destino rd, em nosso banco de registradores.

Figura 14: Operação LWU vista na waveform

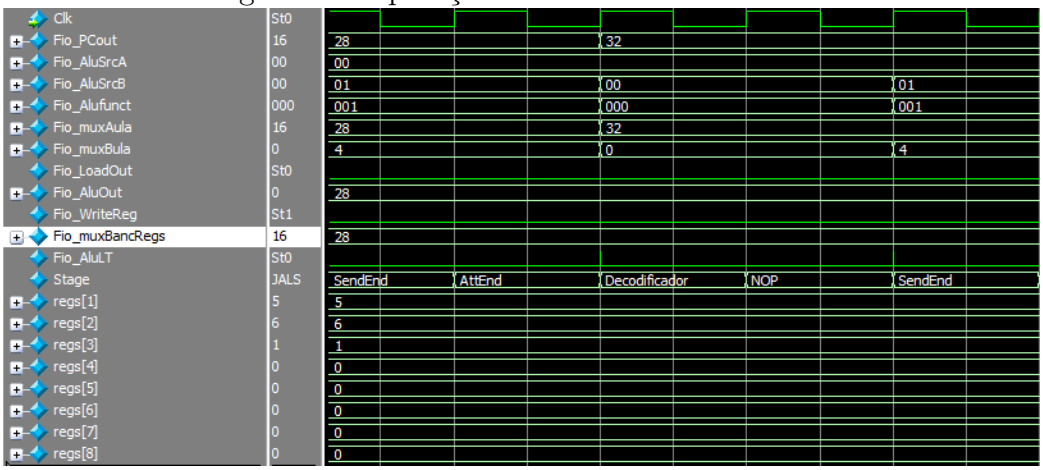


3.2.11 nop

nop *no operation*

Após identificar a instrução nop no estágio de decodificação, voltamos para o estágio de procura de uma nova instrução.

Figura 15: Operação NOP vista na waveform

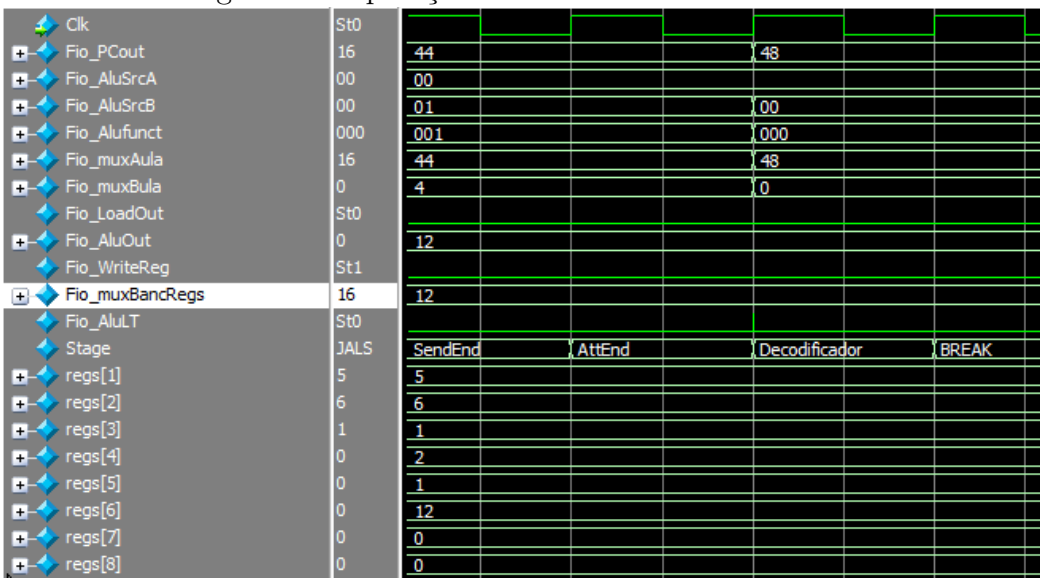


3.2.12 break

break *stop execution*

Após identificar a instrução break no estágio de decodificação, encerramos a procura por novas instruções, ficando em loop infinito no estado BREAK, portanto, parando o processamento de instruções.

Figura 16: Operação BREAK vista na waveform

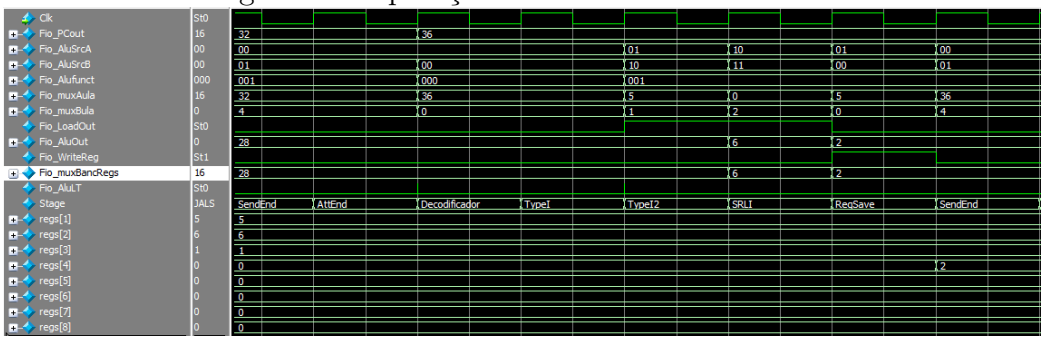


3.2.13 srli

srli rd, rs1, shamt $\quad\quad$ $rd \leftarrow rs1 >> shamt$ *(lógico)*

Após identificar a instrução srli no estágio de decodificação, temos a escrita do registrador rs1 em TempRegA, que posteriormente é passado para o muxE, que controla quem entra no módulo shifter. O valor "shamt"é passado para o módulo de shift indicando quantas vezes o shift será realizado e também na decodificação é indicado qual o tipo de shift que deverá ser realizado, neste caso, o shift right lógico. O valor de TempRegA é shiftado no módulo e está disponível na saída do módulo de shift, que passará para a entrada B da ALU, onde realizaremos uma soma com 0 apenas para ter em AluOut disponível o resultado do nosso shift, que posteriormente será passado e salvo no registrador rd.

Figura 17: Operação SRLI vista na waveform

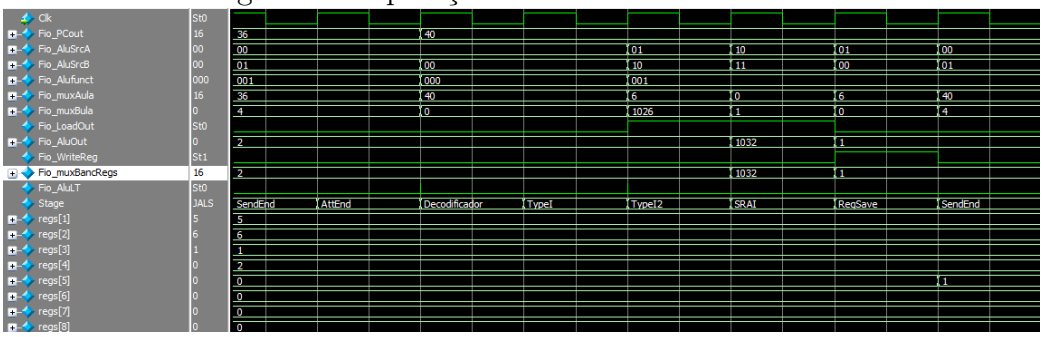


3.2.14 srai

srai rd, rs1, shamt $\quad\quad$ $rd \leftarrow rs1 >> shamt$ *(aritmetico)]*

Após identificar a instrução srai no estágio de decodificação, temos a escrita do registrador rs1 em TempRegA, que posteriormente é passado para o muxE, que controla quem entra no módulo shifter. O valor "shamt"é passado para o módulo de shift indicando quantas vezes o shift será realizado e também na decodificação é indicado qual o tipo de shift que deverá ser realizado, neste caso, o shift right aritmético. O valor de TempRegA é shiftado no módulo e está disponível na saída do módulo de shift, que passará para a entrada B da ALU, onde realizaremos uma soma com 0 apenas para ter em AluOut disponível o resultado do nosso shift, que posteriormente será passado e salvo no registrador rd.

Figura 18: Operação SRAI vista na waveform

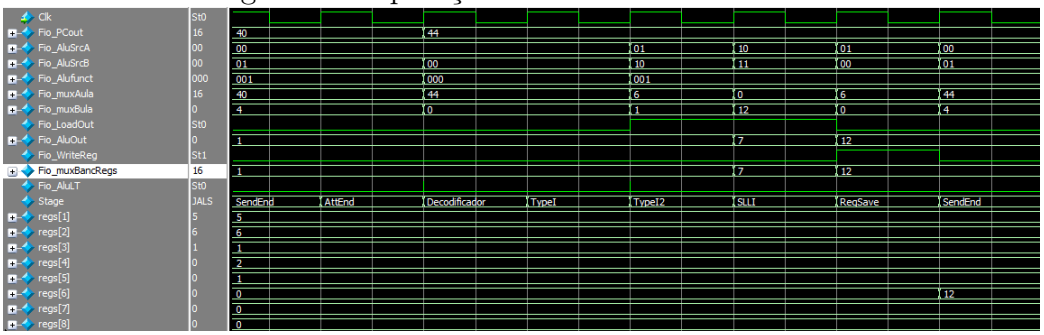


3.2.15 slli

slli rd, rs1, shamt $\qquad$ $rd \leftarrow rs1 << shamt(lógico)$

Após identificar a instrução slli no estágio de decodificação, temos a escrita do registrador rs1 em TempRegA, que posteriormente é passado para o muxE, que controla quem entra no módulo shifter. O valor "shamt"é passado para o módulo de shift indicando quantas vezes o shift será realizado e também na decodificação é indicado qual o tipo de shift que deverá ser realizado, neste caso, o shift left lógico. O valor de TempRegA é shiftado no módulo e está disponível na saída do módulo de shift, que passará para a entrada B da ALU, onde realizaremos uma soma com 0 apenas para ter em AluOut disponível o resultado do nosso shift, que posteriormente será passado e salvo no registrador rd.

Figura 19: Operação SLLI vista na waveform



3.3 Instruções do tipo S

As instruções do tipo S seguem o seguinte formato:

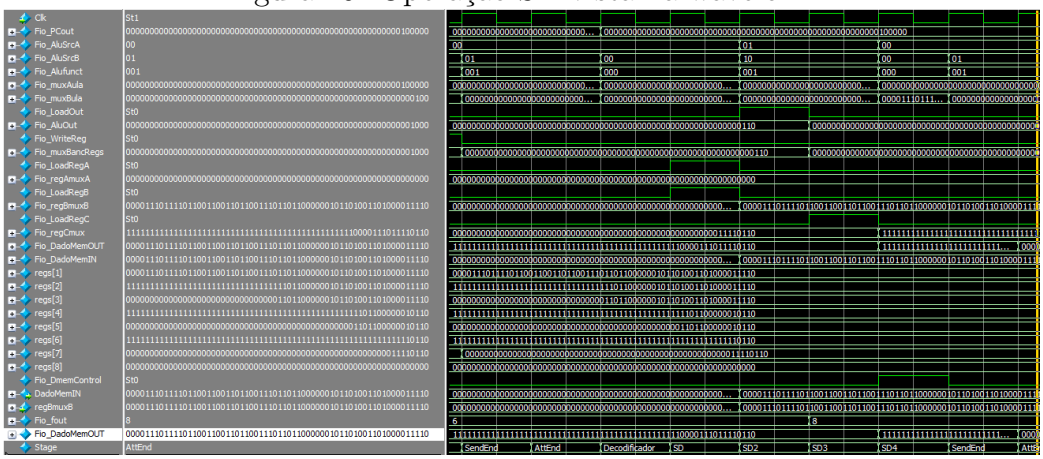
6	5	5	3	5	7
immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode

3.3.1 sd

sd rs2, imm(rs1) *MEM[rs1 + imm] ← rs2*

Após identificar a instrução sd no estágio de decodificação, o valor do registrador rs1 é salvo em TempRegA e o valor do registrador rs2 é salvo em TempRegB. De mesmo modo, o nosso imm passa pelo extensor de sinal, posteriormente passando pelo muxB e indo para a entrada da ALU, junto com o valor de rs1 salvo em TempRegA através do muxA. O sinal alufct é enviado pela unidade de controle indicando que a ALU deverá realizar uma soma. Após a soma, teremos em AluOut o endereço [rs1+imm], que é enviado para o input Address da nossa memória de dados, simultaneamente, o valor de rs2 antes salvo em TempRegB também é passado através do modulo sizeDadoIN, nesse caso sem sofrer alteração para o input writeData da memória, setamos o sinal DmemControl vindo da unidade de controle para escrita e posteriormente teremos salvo em [rs1+imm] o valor de rs2.

Figura 20: Operação SD vista na waveform

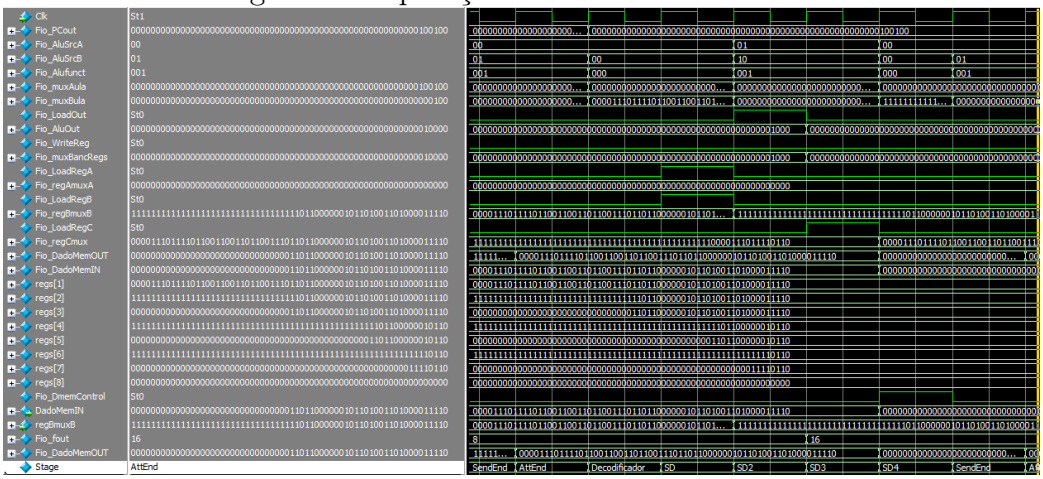


3.3.2 sw

sw rs2, imm(rs1) *MEM[rs1 + imm] ← rs[31:0]*

Após identificar a instrução sw no estágio de decodificação, realizamos um processo de carregamento da memória antes de necessariamente dar o store. Como o nosso processador consegue apenas armazenar de 64 em 64 bits, analogamente consegue apenas carregar de 64 em 64 bits. Então para dar um store em algo do tipo word (32 bits) temos que tomar cuidado para não realizar eventuais sobrescritas. Para tal, passamos nosso imm pelo extensor de sinal, posteriormente passando pelo muxB e indo para a entrada da ALU, junto com o valor de rs1 salvo em TempRegA através do muxA. Realizamos assim uma operação de soma e teremos disponível na saída do registrador AluOut o nosso endereço de destino, cujo o valor atual estará na saida da memória, a qual terá os (32 bits) menos significativos substituidos e depois todo o bloco será carregado na memória, salvando em [rs1 + imm] o valor da word rs2.

Figura 21: Operação SW vista na waveform



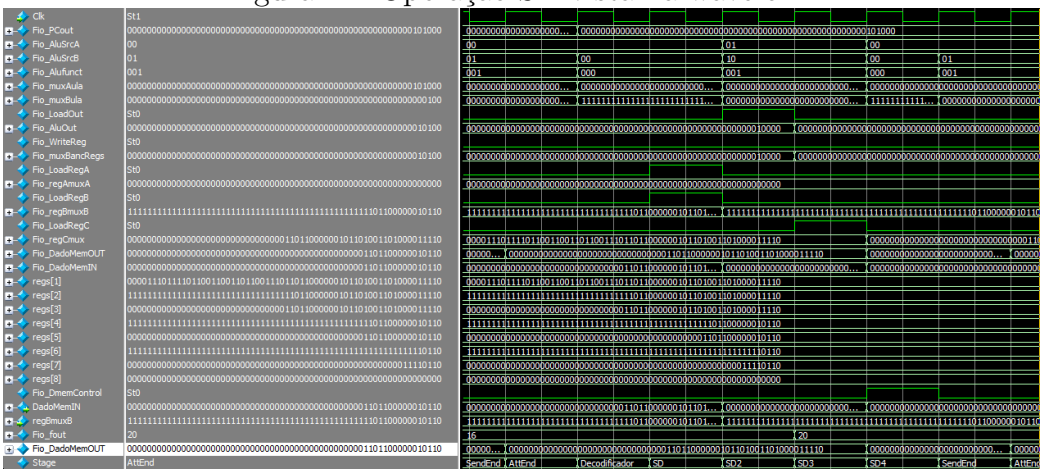
3.3.3 sh

sh rs2, imm(rs1) *MEM[rs1 + imm] ← rs[15:0]*

Após identificar a instrução sh no estágio de decodificação, realizamos um processo de carregamento da memória antes de necessariamente dar o store. Como o nosso processador consegue apenas armazenar de 64 em 64 bits, analogamente consegue apenas carregar de 64 em 64 bits. Então para dar um store em algo do tipo halfword (16 bits) temos que tomar cuidado para não realizar eventuais sobrescritas. Para tal, passamos nosso imm pelo extensor de sinal, posteriormente passando pelo muxB e indo para a entrada da ALU, junto com o valor de rs1 salvo em TempRegA através do muxA.

Realizamos assim uma operação de soma e teremos disponível na saída do registrador AluOut o nosso endereço de destino, cujo o valor atual estará na saida da memória, a qual terá os (16 bits) menos significativos substituidos e depois todo o bloco será carregado na memória, salvando em [rs1 + imm] o valor da halfword rs2.

Figura 22: Operação SH vista na waveform

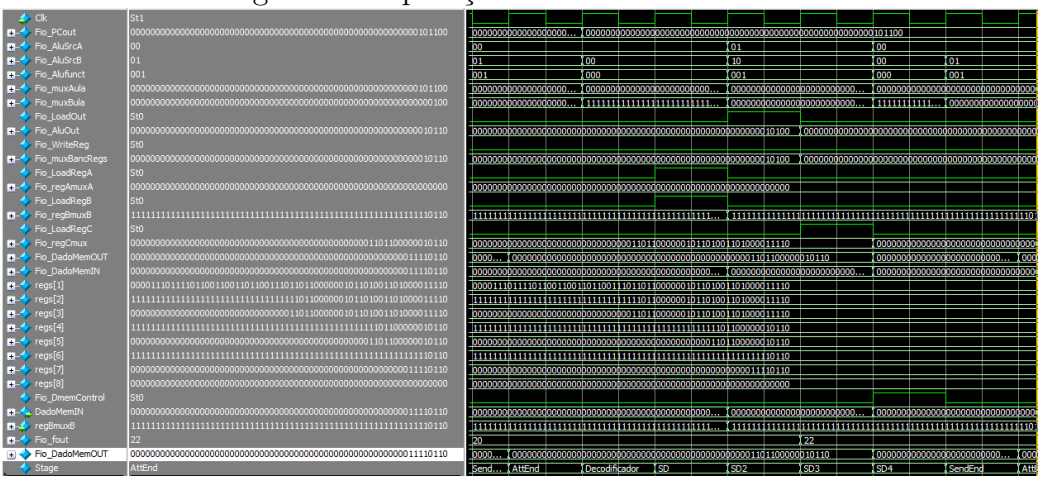


3.3.4 sb

sb rs2, imm(rs1) $\qquad$ $MEM[rs1 + imm] \leftarrow rs[7:0]$

Após identificar a instrução sh no estágio de decodificação, realizamos um processo de carregamento da memória antes de necessariamente dar o store. Como o nosso processador consegue apenas armazenar de 64 em 64 bits, analogamente consegue apenas carregar de 64 em 64 bits. Então para dar um store em algo do tipo byte (8 bits) temos que tomar cuidado para não realizar eventuais sobrescritas. Para tal, passamos nosso imm pelo extensor de sinal, posteriormente passando pelo muxB e indo para a entrada da ALU, junto com o valor de rs1 salvo em TempRegA através do muxA. Realizamos assim uma operação de soma e teremos disponível na saída do registrador AluOut o nosso endereço de destino, cujo o valor atual estará na saida da memória, a qual terá os (8 bits) menos significativos substituidos e depois todo o bloco será carregado na memória, salvando em [rs1 + imm] o valor do byte rs2.

Figura 23: Operação SB vista na waveform



3.4 Instruções do tipo SB

As instruções do tipo SB seguem o seguinte formato:

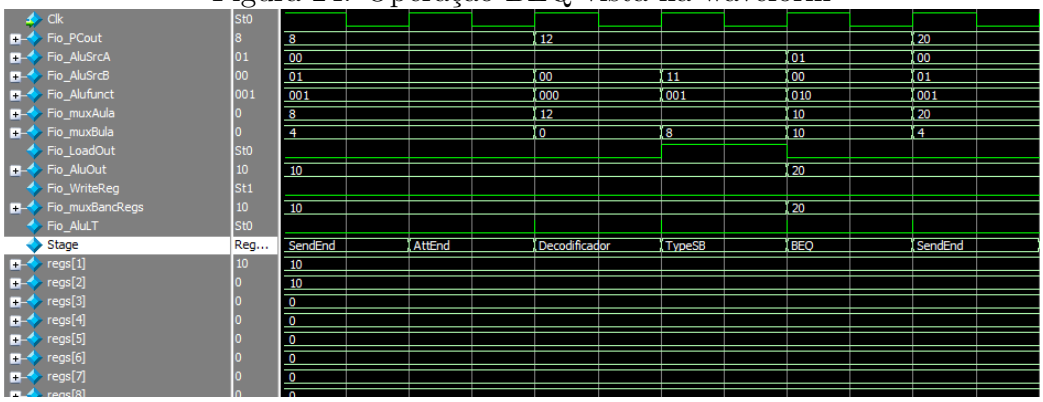
7	5	5	3	5	7
immed[12, 10:5]	rs2	rs1	funct3	immed[4:1, 11]	opcode

3.4.1 beq

beq rs1, rs2, imm $\qquad PC \leftarrow PC + imm*2$, *se rs1 == rs2*

Após identificar a instrução beq no estágio de decodificação, o valor do registrador rs1 é salvo em TempRegA e o valor do registrador rs2 é salvo em TempRegB. Simultaneamente a isto, o nosso imm passa pelo extensor de sinal e pelo shift left, posteriormente passando pelo muxB e também passamos PC pelo muxA, dessa forma podemos calcular PC + deslocamento na ALU e salvar no registrador ALUOut. No ciclo seguinte, fazemos a comparação, para isso utilizando o fio alufct vindo da nossa unidade de controle, usamos a operação de subtração para subtrair o valor de rs1 pelo valor de rs2, caso essa subtração resulte em zero, temos um sinal zero que sai da nossa ALU, responsável por indicar justamente essa igualdade. Em nossa unidade de controle, temos uma verificação do resultado que sai desse sinal zero, de modo que possamos decidir entre manter PC = PC + 4 ou salvar o PC = PC + deslocamento já calculado e salvo no registrador AluOut.

Figura 24: Operação BEQ vista na waveform

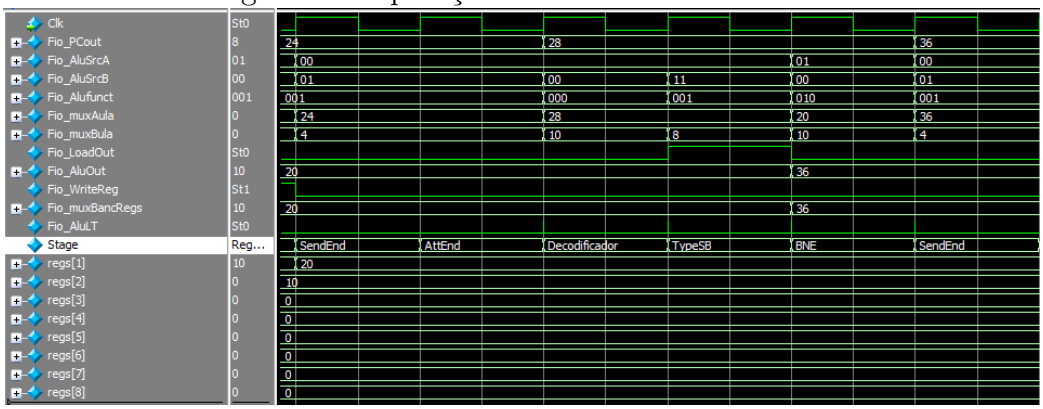


3.4.2 bne

bne rs1, rs2, imm $\qquad PC \leftarrow PC + imm*2$, *se rs1 != rs2*

Após identificar a instrução bne no estágio de decodificação, o valor do registrador rs1 é salvo em TempRegA e o valor do registrador rs2 é salvo em TempRegB. Simultaneamente a isto, o nosso imm passa pelo extensor de sinal e pelo shift left, posteriormente passando pelo muxB e também passamos PC pelo muxA, dessa forma podemos calcular PC + deslocamento na ALU e salvar no registrador ALUOut. No ciclo seguinte, fazemos a comparação, para isso utilizando o fio alufct vindo da nossa unidade de controle, usamos a operação de subtração para subtrair o valor de rs1 pelo valor de rs2, caso essa subtração não resulte em zero, temos um sinal zero que sai da nossa ALU, responsável por indicar justamente essa desigualdade. Em nossa unidade de controle, temos uma verificação do resultado que sai desse sinal zero, de modo que possamos decidir entre manter PC = PC + 4 ou salvar o PC = PC + deslocamento já calculado e salvo no registrador AluOut.

Figura 25: Operação BNE vista na waveform



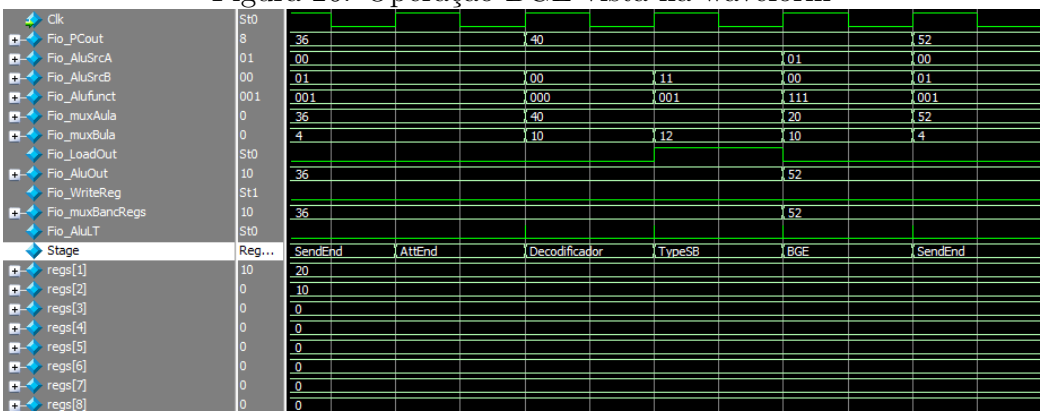
3.4.3 bge

bge rs1, rs2, imm $\qquad PC \leftarrow PC + imm*2$, *se rs1* $\geq$ *rs2*

Após identificar a instrução bne no estágio de decodificação, o valor do registrador rs1 é salvo em TempRegA e o valor do registrador rs2 é salvo em TempRegB. Simultaneamente a isto, o nosso imm passa pelo extensor de sinal e pelo shift left, posteriormente passando pelo muxB e também passamos PC pelo muxA, dessa forma podemos calcular PC + deslocamento na ALU

e salvar no registrador ALUOut. No ciclo seguinte, fazemos a comparação, para isso utilizando o fio alufct vindo da nossa unidade de controle, usamos a operação de comparação de maior igual (>=) para comparar os valores de rs1 e rs2. Dependendo do resultado dessa operação, a unidade de controle escolherá se deverá manter o valor de PC + 4 já calculado no início ou se o sobrescreverá com o valor de PC + descolocamento calculado e salvo no registrador AluOut.

Figura 26: Operação BGE vista na waveform

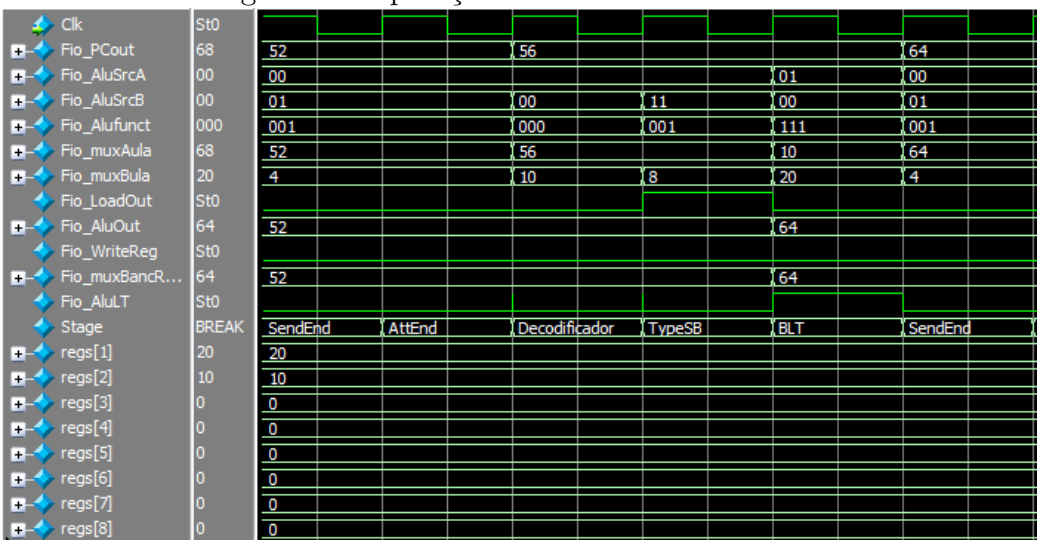


3.4.4 blt

blt rs1, rs2, imm $\qquad PC \leftarrow PC + imm*2$ *, se* $rs1 < rs2$

Após identificar a instrução bne no estágio de decodificação, o valor do registrador rs1 é salvo em TempRegA e o valor do registrador rs2 é salvo em TempRegB. Simultaneamente a isto, o nosso imm passa pelo extensor de sinal e pelo shift left, posteriormente passando pelo muxB e também passamos PC pelo muxA, dessa forma podemos calcular PC + deslocamento na ALU e salvar no registrador ALUOut. No ciclo seguinte, fazemos a comparação, para isso utilizando o fio alufct vindo da nossa unidade de controle, usamos a operação de comparação de maior igual (<) para comparar os valores de rs1 e rs2. Dependendo do resultado dessa operação, a unidade de controle escolherá se deverá manter o valor de PC + 4 já calculado no início ou se o sobrescreverá com o valor de PC + descolocamento calculado e salvo no registrador AluOut.

Figura 27: Operação BLT vista na waveform



3.5 Instruções do tipo U

As instruções do tipo U seguem o seguinte formato:

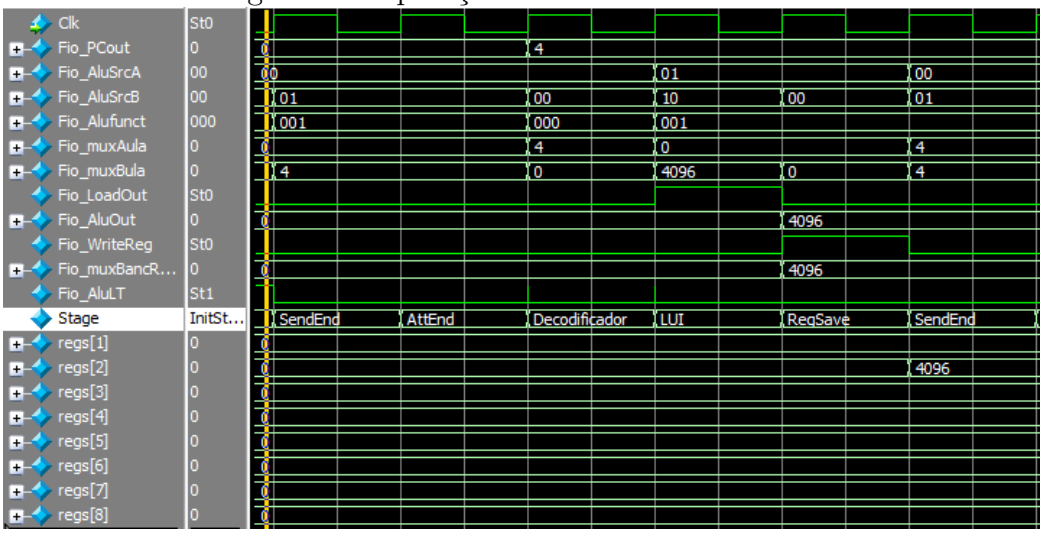
20	5	7
immed[31:12]	rd	opcode

3.5.1 lui

lui rd, imm *rd = 32'b'imm<31>, imm, 12'b0*

Após identificar a instrução lui no estágio de decodificação, resetamos o nosso TempRegA, passamos o nosso imm pelo extensor de sinal e posteriormente pelo muxB. Através do sinal alufct da nossa unidade de controle, realizamos uma soma do nosso imm com TempRegA que foi passado pelo muxA, (com todos os bits zerados). Em AluOut teremos o resultado da nossa soma, que posteriormente será enviada para o nosso banco de registradores e salvo no nosso registrador de destino rd.

Figura 28: Operação LUI vista na waveform



3.6 Instruções do tipo UJ

As instruções do tipo UJ seguem o seguinte formato:

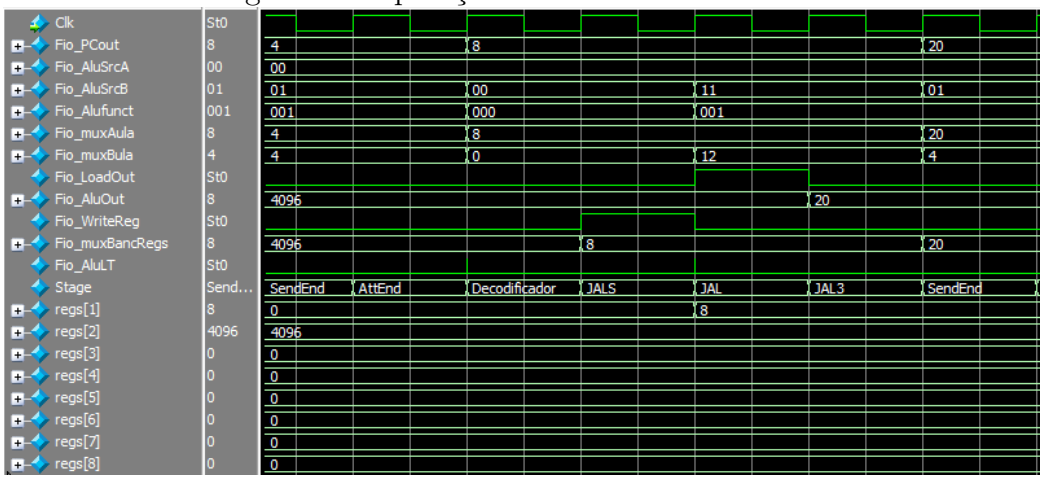
20	5	7
immed[20, 10:1, 11, 19:12]	rd	opcode

3.6.1 jal

jal rd, imm $rd \leftarrow PC;$ $PC \leftarrow PC + (imm[20:1][0])*2$

Após identificar a instrução jal no estágio de decodificação, carregamos o valor de PC (já deslocado + 4) no registrador rd. O valor de imm passa pelo extensor de sinal e depois passa pelo shift (será shiftado uma vez para a esquerda), logo depois pegamos o valor de PC, colocado na saída do muxA (já sendo PC + 4, ou seja, a próxima instrução), somamos na ULA com o imm que vem a partir do muxBula e retornamos essa soma para ser salvo no registrador PC.

Figura 29: Operação JAL vista na waveform



4 Descrição dos Estados do Controle

4.1 initState

O estado InitState é o primeiro estado de funcionamento do processador. Quando ocorre uma situação na qual reset = 1 (que indica que o processador deve ser resetado), a máquina de estados volta para esse estado.

4.2 SendEnd

Nesse estado, temos que o valor de PC é enviado para a ALU, onde ocorre a soma PC + 4 para que ocorra a atualização do endereço da próxima instrução que deve ser buscado na memória de instruções. Nesse mesmo estado o endereço presente em PC também é enviado para que ocorra a busca da instrução que está associada a aquele endereço.

4.3 AttEnd

Nesse estado, o sinal de controle PCWrite recebe o valor 1 (PCWrite = 1) e o muxG tem seu seletor colocado para zero de modo que o registrador PC receba a saída do muxD, com PC+4 permitindo dessa forma que ocorra a atualização do endereço presente no registrador PC. Além disso, o sinal de controle Load_IR recebe o valor 1 (Load_IR = 1), permitindo que a instrução à qual o antigo endereço de PC se referia seja carregada no registrador de instruções(IR).

4.4 Decode

No estado Decode, o campo opcode da instrução é analisado. Além disso, quando necessário, ocorre a análise dos campos func3 e func7. Com base nessa análise, será definido qual o próximo estado que deverá ser iniciado para que o funcionamento do processador continue normalmente.

4.5 TypeR

Nesse estado, os valores presentes nos registradores rs1 e rs2 são carregados para os registradores TempRegA e TempRegB, respectivamente, pois setamos os sinais de controle LoadRegA e LoadRegB para o valor 1.

4.6 Add

Nesse estado, a ALU é setada para realizar a operação de soma (ALUfunc = 3'b001). A entrada "A" da ALU é setada para receber o valor proveniente do registrador TempRegA (AluSrcA = 2'b01) e a entrada "B" da ALU é setada para receber o valor proveniente do registrador TempRegB (AluSrcB = 2'b00). Como a ALU é um componente combinacional, o resultado da operação se encontra disponível nesse mesmo estado e é salvo no registrador AluOut por meio do sinal de controle LoadOut(LoadOut = 1'b1). O sinal de controle MemToReg permanece setado com o valor 0, pois o dado a ser inserido não deve ser aquele proveniente da memória de dados.

4.7 Sub

Nesse estado, a ALU é setada para realizar a operação de subtração (ALUfunc = 3'b010). A entrada "A" da ALU é setada para receber o valor proveniente do registrador TempRegA (AluSrcA = 2'b01) e a entrada "B" da ALU é setada para receber o valor proveniente do registrador TempRegB (AluSrcB = 2'b00). Como a ALU é um componente combinacional, o resultado da operação se encontra disponível nesse mesmo estado e é salvo no registrador AluOut por meio do sinal de controle LoadOut(LoadOut = 1'b1). O sinal de controle MemToReg permanece setado com o valor 0, pois o dado a ser inserido não deve ser aquele proveniente da memória de dados.

4.8 And

Nesse estado, a ALU é setada para realizar a operação de and lógico (ALUfunc = 3'b011). A entrada "A" da ALU é setada para receber o valor proveniente do registrador TempRegA (AluSrcA = 2'b01) e a entrada "B" da ALU é setada para receber o valor proveniente do registrador TempRegB (AluSrcB = 2'b00). Como a ALU é um componente combinacional, o resultado da operação se encontra disponível nesse mesmo estado e é salvo no registrador AluOut por meio do sinal de controle LoadOut(LoadOut = 1'b1). O sinal de controle MemToReg permanece setado com o valor 0, pois o dado a ser inserido não deve ser aquele proveniente da memória de dados.

4.9 RegSave

Nesse estado, o valor que será salvo no registrador de destino não é proveniente da memória de dados. Portanto, o sinal de controle MemToReg é

setado como 0. Além disso, o resultado da operação, que foi salvo no registrador AluOut, é salvo no registrador de destino por meio do sinal de controle WriteReg, que recebe o valor 1. Ademais, não é necessário mais carregar outro valor no registrador AluOut. Logo, LoadOut = 1'b0.

4.10 BEQ

Setamos a ULA para a função de subtração, passamos os nossos registradores previamente carregados da memória para a nossa ULA e usamos o sinal de zero para saber se eles são iguais (caso a - b == 0, significa que são iguais), dessa forma podemos através do MuxC decidir se mantemos o valor de PC + 4 ou se sobrescrevemos com o valor de PC + Deslocamento, para possibilitar isso colocamos o (SrcInst = 1'b0) para o muxG colocar na saída o valor vindo da saida do muxD.

4.11 BNE

Setamos a ULA para a função de subtração, passamos os nossos registradores previamente carregados da memória para a nossa ULA e usamos o sinal negado de zero para saber se eles são diferentes (caso a - b != 0, significa que são diferentes), dessa forma podemos através do MuxC decidir se mantemos o valor de PC + 4 ou se sobrescrevemos com o valor de PC + Deslocamento, para possibilitar isso colocamos o (SrcInst = 1'b0) para o muxG colocar na saída o valor vindo da saida do muxD.

4.12 BLT

Setamos a ULA para a função de comparação (<), passamos os nossos registradores previamente carregados da memória para a nossa ULA e usamos o resultado da ULA (0 caso a comparação seja falsa ou 1 caso a comparação seja verdadeira) para, dessa forma, através do MuxC decidir se mantemos o valor de PC + 4 ou se sobrescrevemos com o valor de PC + Deslocamento, para possibilitar isso colocamos o (SrcInst = 1'b0) para o muxG colocar na saída o valor vindo da saida do muxD.

4.13 BGE

Setamos a ULA para a função de comparação (>=), passamos os nossos registradores previamente carregados da memória para a nossa ULA e usamos o resultado da ULA (0 caso a comparação seja falsa ou 1 caso a comparação seja verdadeira) para, dessa forma, através do MuxC decidir se mantemos o

valor de PC + 4 ou se sobrescrevemos com o valor de PC + Deslocamento, para possibilitar isso colocamos o (SrcInst = 1'b0) para o muxG colocar na saída o valor vindo da saida do muxD.

4.14 SLT

Nesse estado, a ALU é setada para realizar a operação de comparação (ALUFunc = 3'b111). Além disso, a entrada "A"da ALU recebe o valor proveniente do registrador TempRegA (AluSrcA = 2'b01) e a entrada "B"da ALU irá receber o valor proveniente do registrador TempRegB (AluSrcB = 2'b00). A comparação então é feita e caso A ¡ B, teremos que fio_AluLT (de 1 bit) receberá o valor 1. Por outro lado, caso >=, teremos que AluLT receberá o valor 0.

4.15 SLT2

Nesse estado, a Unidade de Controle irá receber o valor proveniente do fio_AluLT. Em seguida, ocorrerá uma extensão de sinal de 1 bit para 64bits e esse valor de 64bits será passado para o fio_flagRD. Nesse mesmo estado, o sinal de controle MemToReg será setado com o valor 2'b11. Isso faz com que o valor que será carregado no registrador de destino seja o valor que está presente em fio_flagRD. Além disso, o sinal de controle WriteReg receberá o valor 1 (WriteReg = 1'b1), permitindo que a escrita desse novo valor no registrador de destino seja efetuada.

4.16 TypeI

Nesse estado, setamos dois sinais de controle da seguinte maneira: LoadRegA = 1'b1 e LoadRegB = 1'b0. Isso faz com que o valor presente no registrador rs1 do banco de registradores seja carregado no registrador temporário A. Entretanto, o mesmo não acontece com o registrador temporário B.

4.17 TypeI2

Nesse estado, a ALU é configurada para realizar a operação de adição (AluFunc = 3'b001). Além disso, a entrada "A"da ALU será o mesmo valor que está contido no registrador TempRegA. Para isso deve-se setar o sinal de controle AluSrcA da seguinte maneira: AluSrcA = 2'b01. Por outro lado, o valor presente na entrada "B"da ALU será o mesmo do imediato presente na instrução que está sendo executada. Isso é garantido setando o sinal de

controle AluSrcB (que seleciona qual o valor deve ir para a entrada "B"da ALU) para o valor 2'b10. Nesse mesmo estado, a operação de rs1 + imm é executada (devido ao caráter combinacional da ALU) e, como LoadOut = 1'b1, temos que tal resultado é salvo no registrador AluOut.

4.18 LType

Nesse estado, ocorre a busca na memória de dados de acordo com o endereço obtido pela soma de rs1 com o imediato. A leitura da memória de dados é autorizada quando o sinal de controle DMemRead recebe o valor 1. Simultaneamente, o sinal de controle LoadOut é setado para o valor 0 e, dessa forma, nenhum valor pode ser sobrescrito no registrador AluOut. Além disso, o sinal de controle MemToReg recebe o valor 2'b01. Esse valor indica que o valor que será escrito no regitrador de destino é proveniente da memória de dados.

4.19 LType2

Nesse estado, ocorre a escrita no registrador TempRegC do dado que foi encontrado (de acordo com o endereço calculado) no estado anterior. É importante observar que nesse estado também ocorre a extensão de sinal do dado para 64 bits. O carregamento do registrador TempRegC é autorizado pelo sinal de controle LoadRegC quando este recebe o valor 1'b1. O sinal de controle MemToReg permanece com o valor 2'b01, assim como no estado anterior.

4.20 LType3

Nesse estado, o valor do registrador TempRegC vai ser escrito no registrador de destino (rd), que se encontra no banco de registradores. O sinal de controle que permite essa escrita é o WriteReg e para que isso aconteça o seu valor deve ser 2'b01. Ademais, o valor presente no sinal de controle MemToReg deve ser 2'b01 para que o valor do TempRegC seja o valor selecionado pelo MuxC para ser escrito no registrador de destino (rd).

4.21 SD

Nesse estado, o que ocorre é o carregamento dos registradores temporários TempRegA e TempRegB. Os sinais de controle que permitem esse carregamento são LoadRegA (que apresenta o valor 1'b1) e LoadRegB (que apresente o valor 1'b1).

4.22 SD2

Nesse estado, a ALU é configurada para realizar a operação de adição (AluFunc = 3'b001). Além disso, a entrada "A"da ALU é configurada para receber o valor presente no registrador TempRegA (AluSrcA = 2'b01). A entrada "B"da ALU é configurada para receber o valor do imediato presente na instrução(AluSrcB = 2'b10). A soma é então realizada e , nesse mesmo, estado o resultado(endereço de destino) é carregado no registrador AluOut (O sinal de controle LoadOut permite a atualização do valor do registrador AluOut quando ele está com o valor 1'b1).

4.23 SD3

Nesse estado, como a memória está sempre lendo, o valor que corresponde ao endereço já calculado previamente e colocado na entrada de endereço da memória, no estado anterior, já se encontra no fio dmemRegC. Combinacionalmente, é feito o "merge"do dado proveniente do TempRegB (rs2) com o dado que está saindo da memória de dados. (Isso é feito devido às variações dos tamanhos das instruções de store) e LoadOut vai para 0, para evitar carregar lixos.

4.24 SD4

O sinal de controle LoadOut recebe o valor 1'b0 e o sinal de controle LoadRegC recebe o valor 1'b0. Além disso, o valor que está salvo no registrador TempRegC deve ser salvo agora na memória de dados. Para isso, o sinal de controle DMemControl recebe o valor 1'b1, permitindo assim a escrita do dado na memória.

4.25 SLTI

Nesse estado, a ALU é configurada para realizar a operação de comparação (AluFunc = 3'b111). Além disso, a entrada "A"da ALU é configurada para receber o valor de TempRegA (AluSrcA = 2'b01) e a entrada "B"da ALU é configurada para receber o valor de TempRegB (AluSrcB = 2'b10).

4.26 NOP

Nesse estado, nenhuma operação é executada. E, logo em seguida, se executa o próximo estado que será o SendEnd.

4.27 BREAK

Uma vez que o processador vai para esse estado, ele ficará permanentemente nele devido a um self-loop.

4.28 JALS

Nesse estado, o sinal de controle MemToReg é setado para o valor 2'b10 (2). Nessa configuração, o valor que irá para o MuxC é o endereço da próxima instrução. O sinal de controle WriteReg é setado para o valor 1'b1, permitindo assim que o endereço da próxima instrução seja carregado no registrador de destino. O sinal de controle LoadRegA é setado para o valor 1'b1, permitindo que o registrador TempRegA seja carregado.

4.29 JALR

Nesse estado, a ALU será configurada para executar a operação de adição (AluFunc = 3'b001). A entrada "A" da ALU é setada para o valor 2'b01 (AluSrcA = 2'b01), indicando que tal entrada receberá o valor proveniente do registrador TempRegA. A entrada "B" da ALU está setada para o valor 2'b10 (AluSrcB = 2'b10), indicando que esta entrada receberá o valor do imediato (já com sinal extendido) presente na instrução. A operação de adição é então realizada e o seu resultado é carregado no registrador AluOut (LoadOut = 1'b1).

4.30 JAL

Nesse estado, a ALU é configurada para realizar a operação de adição (AluFunc = 3'b001). A entrada "A" da ALU é configurada para receber o valor proveniente do registrador PC (AluSrcA = 2'b00). A entrada "B" da ALU é configurada para receber o valor do imediato (que já passou pelo processo de extensão de sinal) e que também já sofreu um shift-left (devido ao sinal de controle ShiftSrc que é setado com o valor 1'b1). O sinal de controle responsável por selecionar tal entrada é o AluSrcB (que é setado com o valor 2'b11). A operação de adição é então efetuada e o seu resultado é salvo no registrador AluOut (LoadOut = 1'b1).

4.31 JAL3

Nesse estado, o sinal de controle PCSource é setado para o valor 1'b1, indicando que o valor presente no registrador AluOut é o que deve passar

pelo MuxD para ser escrito no registrador PC. Além disso, o sinal de controle PCWrite é setado para o valor 1'b1, permitindo assim a atualização do registrador PC.

4.32 SLLI

Nesse estado, a ALU é configurada para realizar a operação de adição (AluFunc = 3'b001). A entrada "A" da ALU é configurada para receber o valor 0 (AluSrcA = 2'b10). A entrada "B" da ALU é configurada para receber o valor que se encontra no módulo Shifter (AluSrcB = 2'b11). O sinal de controle ShiftSrc (que é setado com o valor 1'b0) vai deixar passar o valor presente no registrador TempRegA para o módulo Shifter (no qual esse valor irá de fato sofrer a operação de shift). O sinal de controle ShiftControl é setado para o valor 2'b00, que define que o shift a ser executado será um shift-left lógico. O sinal de controle Nshift por sua vez indica quantos shifts serão executados. A operação de Shift é então executada e o seu resultado é carregado no registrador AluOut (após a soma com 0 na ALU) por meio do sinal de controle LoadOut, que está setado com o valor 1'b1.

4.33 SRAI

Nesse estado, a ALU é configurada para realizar a operação de adição (AluFunc = 3'b001). A entrada "A" da ALU é configurada para receber o valor 0 (AluSrcA = 2'b10). A entrada "B" da ALU é configurada para receber o valor que se encontra no módulo Shifter (AluSrcB = 2'b11). O sinal de controle ShiftSrc (que é setado com o valor 1'b0) vai deixar passar o valor presente no registrador TempRegA para o módulo Shifter (no qual esse valor irá de fato sofrer a operação de shift). O sinal de controle ShiftControl é setado para o valor 2'b10, que define que o shift a ser executado será um shift-right aritmetico. O sinal de controle Nshift por sua vez indica quantos shifts serão executados. A operação de Shift é então executada e o seu resultado é carregado no registrador AluOut (após a soma com 0 na ALU) por meio do sinal de controle LoadOut, que está setado com o valor 1'b1.

4.34 SRLI

Nesse estado, a ALU é configurada para realizar a operação de adição (AluFunc = 3'b001). A entrada "A" da ALU é configurada para receber o valor 0 (AluSrcA = 2'b10). A entrada "B" da ALU é configurada para receber o valor que se encontra no módulo Shifter (AluSrcB = 2'b11). O sinal de controle ShiftSrc (que é setado com o valor 1'b0) vai deixar passar o

valor presente no registrador TempRegA para o módulo Shifter (no qual esse valor irá de fato sofrer a operação de shift). O sinal de controle ShiftControl é setado para o valor 2'b01, que define que o shift a ser executado será um shift-right lógico. O sinal de controle Nshift por sua vez indica quantos shifts serão executados. A operação de Shift é então executada e o seu resultado é carregado no registrador AluOut (após a soma com 0 na ALU) por meio do sinal de controle LoadOut, que está setado com o valor 1'b1.

4.35 LUI

Nesse estado, primeiramente o registrador TempRegA tem o seu conteúdo resetado por meio da atuação do sinal de controle resetRegA (resetRegA = 1'b1). O sinal de controle AluSrcB é setado para o valor 2'b10 (AluSrcB = 2'b10). Dessa maneira o valor da entrada "B" da ALU será o valor do imediato presente na instrução já com o sinal extendido. O sinal de controle AluSrcA é setado para o valor 2'b01, definindo que a entrada "A" da ALU receba o valor presente no registrador TempRegA. A ALU é configurada para realizar a operação de adição (AluFunc = 3'b001) e, em seguida, o resultado dessa operação é carregado no registrador AluOut devido a ação do sinal de controle LoadOut, que é setado para o valor 1'b1, permitindo assim esse carregamento.

4.36 TypeSB

Nesse estado, os registradores TempRegA e TempRegB são carregados, devido a atuação dos sinais de controle LoadRegA e LoadRegB (ambos são setados para o valor 1'b1, permitindo assim o carregamento desses registradores). Os valores presentes nesses dois registradores são os valores que serão comparados, afim de checar se o desvio ocorrerá ou não. A entrada "A" da ALU é configurada para receber o valor de PC (AluSrcA = 2'b00). A entrada "B", por sua vez, é configurada para receber o valor proveniente do módulo Shifter. O sinal de controle ShiftSrc é setado para o valor 1'b1, definindo que o valor a ser sofrer o shift-left lógico uma única vez é o valor do imediato (já com sinal extendido), presente na instrução. A ALU então é configurada para realizar a operação de adição (AluFunc = 3'b001). A soma entre PC e o imediato é então realizada e o resultado é carregado no registrador AluOut por ação do sinal de controle LoadOut (LoadOut = 1'b1).

4.37 Overflow

Nesse estado carregamos no registrador EPC o endereço PC - 4 e também carregamos no registrador de causa o motivo da exceção. Simultaneamente, carregamos o endereço 254 da memória de instrução, que estará disponível no estado JMPC, já estendida com zeros pelo sizeDadoOut, por conta da saída exception da UC que é colocada para um, pronto para ser carregado no PC atual e ir para o endereço onde será tratada a exceção de overflow.

4.38 NOpcode

Nesse estado carregamos no registrador EPC o endereço PC - 4 e também carregamos no registrador de causa o motivo da exceção. Simultaneamente, carregamos o endereço 255 da memória, que estará disponível no estado JMPC, já estendida com zeros pelo sizeDadoOut, por conta da saída exception da UC que é colocada para um, pronto para ser carregado no PC atual e ir para o endereço onde será tratada a exceção de NOpcode.

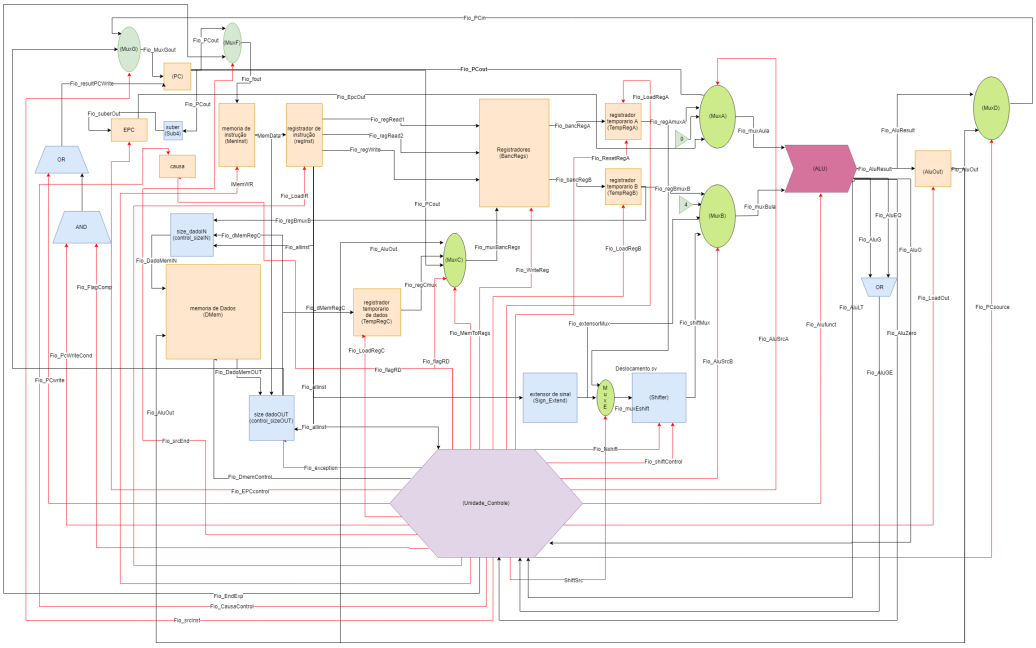
4.39 JMPC

Nesse estado teremos disponível o endereço do tratamento de exceção carregado previamente da memória, que deve ser enviado para ser substituído em PC. Para isso, setamos o muxG para 1, de modo a passar o valor carregado da memória e também setamos pcWrite para 1, para escrever em PC o nosso endereço do tratamento de exceção.

5 Anexos

5.1 Diagrama Estrutural

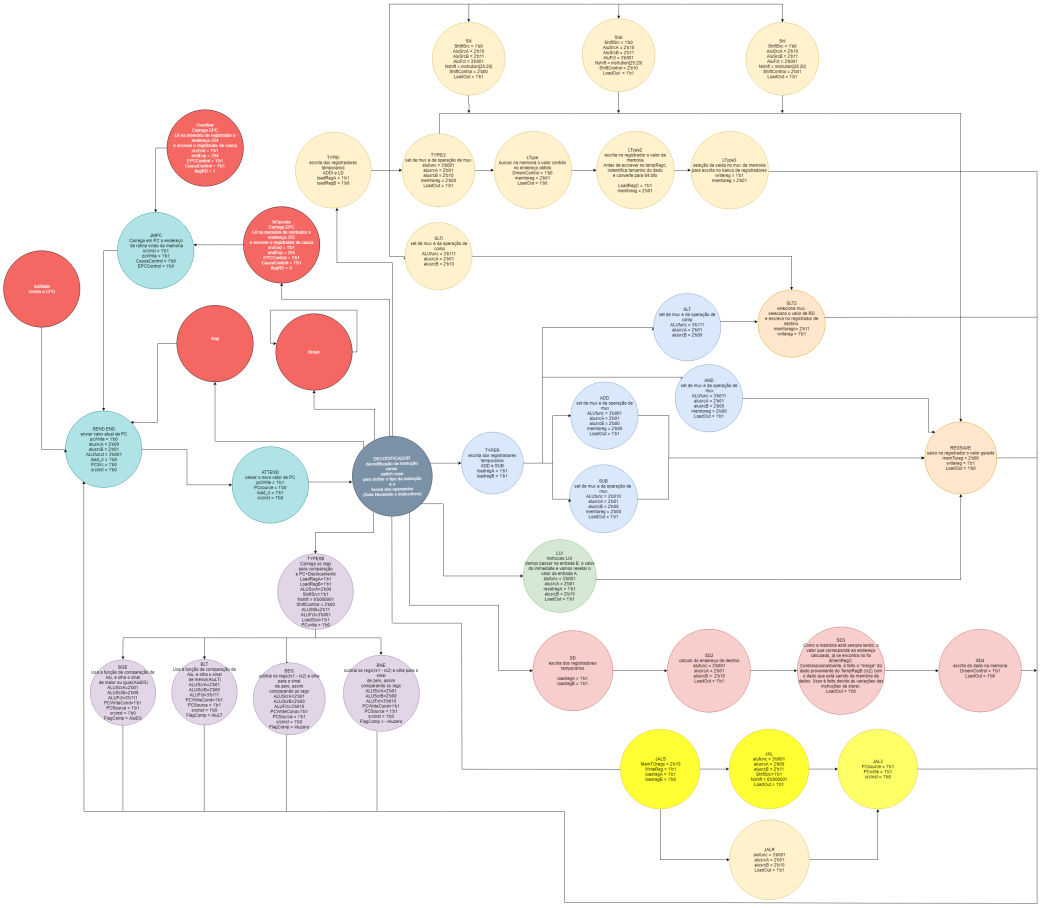
Figura 30: Diagrama Estrutural



Link para acesso em pdf: https://bit.ly/2GJjVi6

5.2 Diagrama de Estados

Figura 31: Diagrama de Estados



Link para acesso em pdf: https://bit.ly/2LbkdDJ

6 Conclusão

A maior dificuldade do projeto foi trabalhar com os stores (sb, sh, sw), porque o processador apenas dava store em 64 bits e tivemos que fazer um load para salvar os 64 bits do endereço especificado e apenas modificar os bits necessários para só após isso dar um store de 64 bits sem alterar os bits que existiam anteriormente.

Outra dificuldade para a realização do projeto foi o tratamento de exceções. Embora a implementação do tratamento de exceções não seja complexa, o entendimento da sua lógica mostrou-se complicado e demorado.

Conclui-se que o projeto foi muito gratificante, uma vez que a sua realização proporcionou um novo entendimento do funcionamento do processador RISC-V. Em sala de aula, o processador foi estudado puramente de um ponto de vista teórico, baseando-se no livro-texto utilizado na disciplina. Entretanto, com o projeto foi possível entender o processador RISC-V de um ponto de vista prático.

Ademais, no decorrer do projeto, os alunos puderam aprender mais sobre linguagem de descrição de hardware (no caso, foi utilizado systemverilog no projeto).