Universidade Federal de Pernambuco
Centro de Informática - CIn

RELATÓRIO DO PROJETO

Abhner Adriel Cristóvão Silva — aacs2
Gabriel Monteiro Silva — gms2
Vinícius Vieira Moreira — vvm
Yano Rodrigues Vasconcelos — yrv

01/07/2025

Índice

Introdução

Este relatório técnico detalha o projeto e a implementação de uma Unidade Central de Processamento (CPU) desenvolvida no âmbito da disciplina. O documento está estruturado para fornecer uma compreensão abrangente da arquitetura da CPU e de seus elementos constituintes.

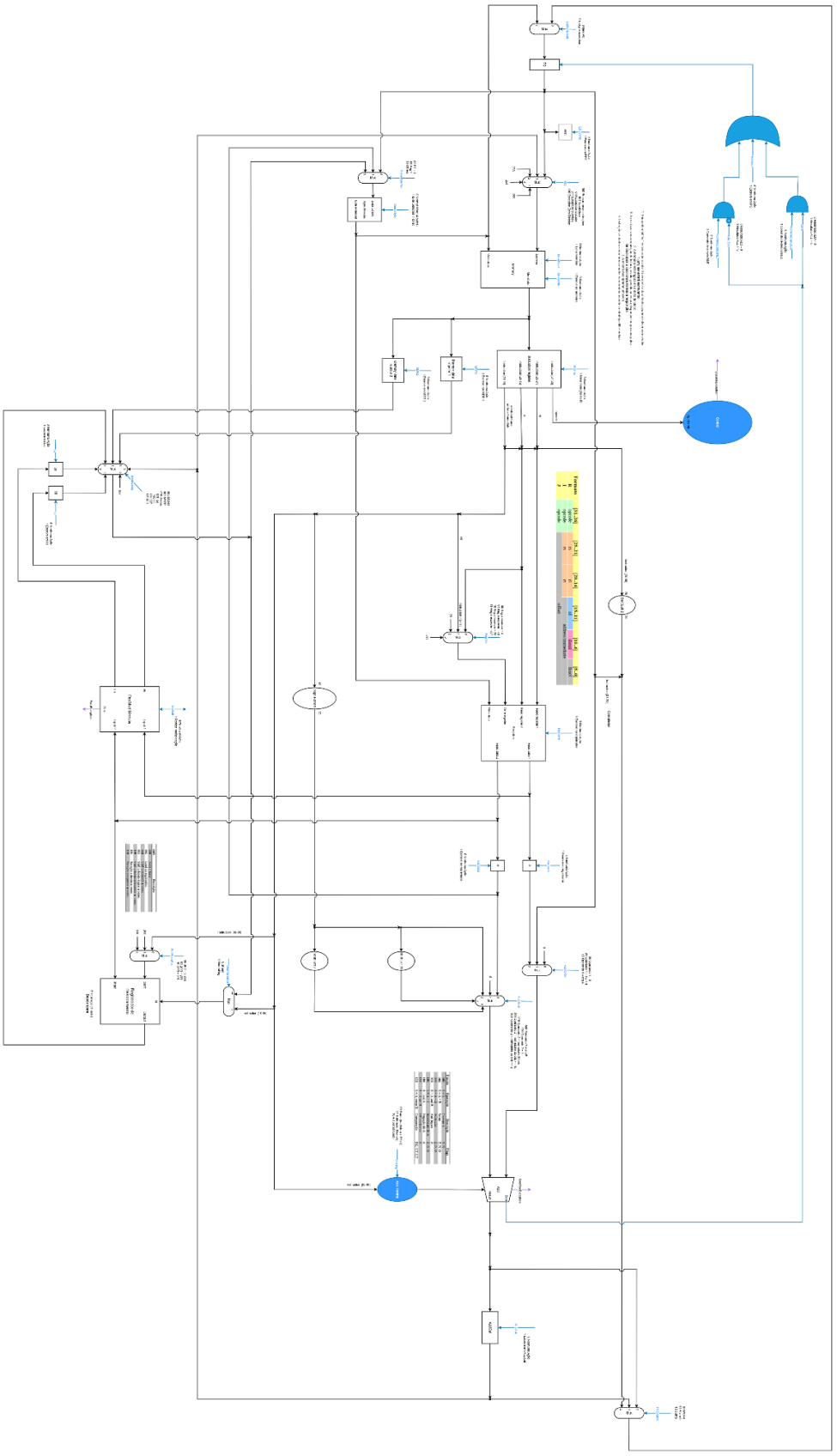
A seção "Unidade de Processamento" apresentará o diagrama de blocos completo da CPU, ilustrando a interconexão e o fluxo de dados entre os principais módulos que a compõem. Este diagrama servirá como uma representação visual da arquitetura global do processador, incluindo a Unidade de Controle, a Unidade Lógica e Aritmética (ULA), o Banco de Registradores, a Memória, e outros componentes essenciais.

Subsequentemente, a seção "Descrição das Entidades" aprofundará na descrição de cada componente individualmente projetado. Serão detalhadas as funcionalidades, entradas e saídas de módulos como a Unidade de Controle, a ULA, os multiplexadores, os registradores, e as unidades de deslocamento e extensão de sinal.

A seção "Descrição dos Estados do Controle" abordará exaustivamente a máquina de estados finitos implementada na Unidade de Controle. Serão detalhados os diferentes estados operacionais, como busca de instrução (ST_READ_F_MEMORY), decodificação de instrução (ST_INTRUCTION_DECODE), e os estados específicos para a execução de cada tipo de instrução (ex: ST_ADD, ST_ADDI, ST_MFLO, ST_MFHI, ST_LB), explicando as transições entre eles e os sinais de controle gerados em cada etapa.

Por fim, a seção "Conjunto de Simulações" apresentará os resultados obtidos de algumas simulações dos componentes desenvolvidos. É importante ressaltar que, devido a limitações no processo de desenvolvimento, não foi possível implementar a totalidade do conjunto de instruções desejado para simulação completa da CPU. No entanto, as simulações existentes demonstram a funcionalidade e o comportamento esperado dos módulos isolados e suas interações básicas.

Unidade de Processamento



Acessível pelo link: [Unidade de Processamento]

Descrição das Entidades

Unidade: Extensor de Sinal de 16 bits para 32 bits

Entradas:

1. Input (16 bits) - vetor de bits a ser estendido

Saídas:

1. Output (32 bits) - vetor de bits estendido em 16 bits (total de 32 bits)

Objetivos: Unidade criada para estender, com sinal, os sinais de 16 bits (endereço ou imediato) das operações do tipo I.
Algoritmo: Caso o bit mais significativo (sinal) seja 0 (positivo), o componente adiciona 0s a esquerda, totalizando 32 bits e mantendo o sinal positivo, caso o número representado seja negativo (bit mais significativo 1), o número é preenchido com 1s, mantendo também o sinal e o valor.

Unidade: Byte Selector

Entradas:

1. ctrl (1 bit) - sinal de controle
2. Input (32 bits) - vetor de bits que irá passar pelo byte selector inteiro ou parcial

Saídas:

1. Output (32 bits) - sinal após a seleção, pode ser 32 bits de fato (whole word) ou apenas os 8 bits menos significativos de Input, tendo os outros como 0.

Objetivos: Unidade criada para a implementação das instruções load byte e store byte. Este componente realiza a seleção de propagar o sinal original de 32 bits ou apenas 8 bits, a depender da instrução, na forma de sinal de controle.
Algoritmo: Se ctrl igual a 1, propaga Input, se não, seleciona os 8 bits menos significativos.
Seguindo a estrutura solicitada, apresento a explicação dos componentes dos arquivos anexados:

Unidade: ShiftLeft2top

Entradas:

1. di1 (5 bits) - Vetor de bits de entrada
2. di2 (5 bits) - Vetor de bits de entrada
3. di3 (16 bits) - Vetor de bits de entrada

Saídas:

1. do (28 bits) - Vetor de bits resultante do deslocamento à esquerda

Objetivos: Esta unidade é utilizada para concatenar três entradas de diferentes tamanhos (di1, di2, di3) e, em seguida, deslocar o resultado concatenado 2 bits para a esquerda.
Algoritmo: As três entradas (di1, di2, di3) são concatenadas na ordem {di1, di2, di3}. O resultado da concatenação é então deslocado 2 bits para a esquerda.

Unidade: control_unit

Entradas:

1. clk (1 bit) - Sinal de clock
2. reset (1 bit) - Sinal de reset
3. OverflowException (1 bit) - Flag de exceção de estouro
4. ng (1 bit) - Flag (indefinida na descrição, mas presente)
5. Zero (1 bit) - Flag de resultado zero da ULA
6. ZeroException (1 bit) - Flag de exceção de divisão por zero
7. eq (1 bit) - Flag de igualdade
8. gt (1 bit) - Flag de maior que
9. lt (1 bit) - Flag de menor que
10. opcode (6 bits) - Parte do código da instrução
11. funct (6 bits) - Campo funct da instrução (para instruções tipo R)

Saídas:

1. PCWrite (1 bit) - Habilita a escrita no Program Counter
2. EPCWrite (1 bit) - Habilita a escrita no Exception Program Counter
3. ALUOp (2 bits) - Seleciona a operação da ALU
4. MemWrite (1 bit) - Habilita a escrita na memória
5. IRWrite (1 bit) - Habilita a escrita no Instruction Register
6. RegWrite (1 bit) - Habilita a escrita no banco de registradores
7. ControlA (1 bit) - Sinal de controle para o registrador A
8. ControlB (1 bit) - Sinal de controle para o registrador B
9. ALUOut (1 bit) - Habilita a escrita no registrador ALUOut
10. DivMult (1 bit) - Sinal de controle para a unidade de divisão/multiplicação
11. Hi (1 bit) - Habilita a escrita no registrador Hi
12. Lo (1 bit) - Habilita a escrita no registrador Lo
13. PCWriteCondBEQ (1 bit) - Habilita a escrita condicional no PC para BEQ
14. PCWriteCondBNE (1 bit) - Habilita a escrita condicional no PC para BNE
15. MDR1 (1 bit) - Habilita a escrita no registrador MDR1
16. MDR2 (1 bit) - Habilita a escrita no registrador MDR2
17. IorD (3 bits) - Seleciona a origem do endereço para a memória (Instrução ou Dados)
18. ContOrExcep (1 bit) - Seleciona entre PC normal ou endereço de exceção
19. RegDst (2 bits) - Seleciona o registrador de destino para escrita
20. ALUSrcA (2 bits) - Seleciona a primeira entrada da ALU
21. ALUSrcB (3 bits) - Seleciona a segunda entrada da ALU
22. PCSource (2 bits) - Seleciona a próxima fonte do PC
23. ShamtSource (1 bit) - Seleciona a fonte para o campo shamt
24. MemToReg (3 bits) - Seleciona a origem dos dados a serem escritos no registrador
25. ShiftFuncSrc (2 bits) - Seleciona a função de deslocamento
26. SelectByteSrc (2 bits) - Seleciona a fonte para o byte selector
27. SelectByte (1 bit) - Habilita a seleção de byte na unidade Byte Selector

Objetivos: A control_unit é o cérebro da CPU, responsável por gerar todos os sinais de controle necessários para o correto funcionamento dos demais componentes. Ela opera como uma máquina de estados finitos, transicionando entre diferentes estados para executar as fases de busca, decodificação e execução das instruções.

Algoritmo: No reset, a unidade entra no estado ST_RESET e inicializa todos os sinais de controle para 0, além de setar resett para 1. Após o reset, a unidade transita para ST_READ_F_MEMORY, onde configura os sinais para buscar a instrução da memória e incrementar o PC. No estado ST_INTRUCTION_DECODE, a instrução é decodificada e o próximo estado é determinado com base no opcode e funct da instrução. Para cada tipo de

instrução (R-type, I-type, J-type), a máquina de estados possui estados específicos que configuram os sinais de controle de forma apropriada para a execução daquela instrução. Exemplo: Para a instrução ADD, no COUNTER == 5'd4, ALUOp é definido como 2'b10, ALUOut como 1'b1, RegDst como 2'b01 e ALUSrcA como 2'b10. No COUNTER == 5'd5, MemToReg é 3'b110, SelectByteSrc é 2'b10 e RegWrite é 1'b1, após isso o reset é ativado para a próxima instrução. A unidade utiliza um contador (COUNTER) para gerenciar as subtarefas dentro de cada estado principal.

Unidade: ShiftLeft16

Entradas:

1. di (32 bits) - Vetor de bits de entrada

Saídas:

1. do (32 bits) - Vetor de bits resultante do deslocamento à esquerda

Objetivos: Esta unidade tem como objetivo realizar um deslocamento lógico de 16 bits para a esquerda em uma entrada de 32 bits.
Algoritmo: A saída do é o resultado do deslocamento de di em 16 posições para a esquerda (di << 16).

Unidade: Sign_extend_16

Entradas:

1. data_in (16 bits) - Vetor de bits a ser estendido

Saídas:

1. data_out (31 bits) - Vetor de bits estendido com sinal (total de 32 bits)

Objetivos: Unidade criada para estender, com sinal, os sinais de 16 bits (endereço ou imediato) das operações do tipo I.
Algoritmo: Se o bit mais significativo de data_in (o bit de sinal, data_in[15]) for 1 (indicando um número negativo), data_out é preenchido com 16 bits '1' seguidos pelos 16 bits de data_in. Caso contrário (se data_in[15] for 0, indicando um número positivo), data_out é preenchido com 16 bits '0' seguidos pelos 16 bits de data_in.

Unidade: ShiftLeft2bottom

Entradas:

1. di (32 bits) - Vetor de bits de entrada

Saídas:

1. do (32 bits) - Vetor de bits resultante do deslocamento à esquerda

Objetivos: Esta unidade realiza um deslocamento lógico de 2 bits para a esquerda em uma entrada de 32 bits.
Algoritmo: A saída do é o resultado do deslocamento de di em 2 posições para a esquerda (di << 2).

Unidade: Cpu

Entradas:

1. clk (1 bit) - Sinal de clock
2. reset (1 bit) - Sinal de reset

Saídas: (Saídas de interconexão entre os componentes internos, não externas ao módulo principal).
Objetivos: O módulo Cpu é a representação do processador completo, integrando todas as unidades funcionais (registrador de programa, banco de registradores, ALU, memória, unidade de controle, etc.) para executar as instruções de um processador. Ele orquestra o fluxo de dados e controle entre esses componentes
Algoritmo: A Cpu instancia e conecta todos os submódulos necessários para seu funcionamento. Inclui registradores como regPC, regEPC, regMDR1, regMDR2, regLO, regHI, regA, regB, regALUOUT, regDESLOC. Possui um banco de registradores (Banco_reg) e uma unidade de memória (Memória). Integra a ula32 (ALU), SignExtend, ShiftLeft16, ShiftLeft2bottom, ShiftLeft2top, alu_control e control_unit. Utiliza múltiplos multiplexadores (Mux_1bit, Mux_2bits, Mux_3bits, Mux_muxshamtsource, Mux_muxshiftfuncsrc, Mux_muxpcsource) para rotear os dados e sinais de controle. Define constantes como c_255, c_254, c_253, c_29, c_r31, c_227, c_0, c_4, c_001, c_010 para uso interno. A control_unit (instanciada como control) recebe as entradas da CPU (clock, reset, flags da ALU) e as partes da instrução (opcode, funct) para gerar os sinais de controle que dirigem o fluxo de dados e as operações dos demais componentes.

Unidade: alu_control

Entradas:

1. ctrl (2 bits) - Sinal de controle principal para a ALU
2. o0 (6 bits) - Campo funct da instrução (relevante quando ctrl indica operação tipo R)

Saídas:

1. out (3 bits) - Código da operação a ser executada pela ULA

Objetivos: A alu_control é responsável por traduzir os sinais de controle recebidos da unidade de controle e o campo funct da instrução em um código de 3 bits que a Unidade Lógica e Aritmética (ULA) pode interpretar para realizar a operação correta (ADD, AND, SUB, SLT, JR, XCHG, etc.).
Algoritmo: Se ctrl é 2'b00, a saída out é 3'b001 (provavelmente ADD ou uma operação base). Se ctrl é 2'b01, a saída out é 3'b010 (provavelmente SUB ou outra operação base). Se ctrl é 2'b10, a alu_control usa o campo o0 (funct) para determinar a operação específica: ADD: out é 3'b001. AND: out é 3'b011. SLT: out é 3'b111. JR: out é 3'b000. XCHG: out é 3'b000. SUB: out é 3'b010. Para outros valores de o0, a saída padrão é 3'b000.

Descrição dos Estados do Controle

Estado: Reset (Implícito no `resett == 1'b1`)

Quando o sinal `resett` está ativo, todos os sinais de controle do processador são inicializados para seus valores padrão. Este é um estado de segurança para garantir um início limpo da operação. Após a inicialização, o estado transita para `ST_READ_F_MEMORY`. Os sinais para escrita no PC, escrita no IR e as fontes da ALU são configurados para o início da busca de instrução.

Estado: ST_READ_F_MEMORY

Este é o estado inicial de busca de instrução. O processador busca a instrução da memória e incrementa o contador de programa (PC). O próximo estado marca o fim da fase de busca de instrução, indicando que a instrução foi carregada e o processador está pronto para decodificá-la. O estado transita para `ST_INTRUCTION_DECODE`, preparando os registradores de controle A e B e as entradas da ALU para a próxima fase.

Estado: ST_INTRUCTION_DECODE

Neste estado, a instrução lida da memória é decodificada. O `opcode` e o `funct` (para instruções R-type) são analisados para determinar o tipo de operação a ser realizada. Com base nessa decodificação, o processador transiciona para o estado de execução específico da instrução. O próximo estado indica o momento em que a decodificação da instrução está completa, permitindo que a unidade de controle determine o fluxo de execução subsequente com base nos campos `opcode` e `funct` da instrução. Instruções não reconhecidas ativam um reset.

Estado: ST_ADD_SUB_AND

Este estado executa operações aritméticas e lógicas do tipo R (ADD, SUB, AND). O próximo estado inicia a fase de execução da operação R-type, configurando a ULA para a operação apropriada e habilitando a escrita do resultado da ULA. O próximo estado finaliza a execução da instrução, direcionando o resultado da ULA para o registrador de destino no banco de registradores e ativando a escrita no mesmo, seguido de um reset para a próxima instrução.

Estado: ST_ADDI

Este estado executa a instrução ADDI (Add Immediate). O próximo estado inicia o cálculo do endereço/valor da instrução ADDI, configurando as entradas da ULA. O próximo estado finaliza a escrita do resultado no registrador de destino, após o cálculo da ULA, e prepara o processador para a próxima instrução.

Estado: ST_DIV

Este estado gerencia a operação de divisão. O próximo estado marca o início da operação de divisão, ativando o sinal de controle para a unidade de divisão/multiplicação. O próximo estado indica a conclusão da operação de divisão, habilitando a escrita nos registradores `Hi` e `Lo` com os resultados e preparando para o próximo ciclo de instrução. O estado “intermediário” garante que o contador continue incrementando durante os ciclos de execução da divisão.

Estado: ST_MFLO

Este estado executa a instrução MFLO (Move from LO). O próximo estado indica o momento de transferir o conteúdo do registrador `Lo` para o registrador de destino, finalizando a instrução.

Estado: ST_MFHI

Este estado executa a instrução MFHI (Move from HI). O próximo estado indica o momento de transferir o conteúdo do registrador `Hi` para o registrador de destino, finalizando a instrução.

Estado: ST_LB

Este estado executa a instrução LB (Load Byte). O próximo estado inicia o cálculo do endereço efetivo na memória para a operação de carga de byte. O próximo estado seleciona a memória de dados como fonte para a operação de leitura de byte. O próximo estado finaliza a carga do byte, transferindo o dado lido para o MDR1, selecionando o byte correto e habilitando a escrita no registrador de destino. O estado “intermediário” assegura que o contador continue avançando nos ciclos intermediários de leitura.

Estado: ST_LW

Este estado executa a instrução LW (Load Word). O próximo estado inicia o cálculo do endereço efetivo na memória para a operação de carga de palavra. O próximo estado seleciona a memória de dados como fonte para a operação de leitura de palavra. O próximo estado finaliza a carga da palavra, transferindo o dado lido para o MDR1 e habilitando a escrita no registrador de destino. O estado “intermediário” assegura que o contador continue avançando nos ciclos intermediários de leitura.

Estado: ST_LUI

Este estado executa a instrução LUI (Load Upper Immediate). O próximo estado inicia o processamento do imediato, configurando a ULA para posicioná-lo na parte superior do registrador. O próximo estado finaliza a escrita do resultado no registrador de destino, após o processamento do imediato.

Estado: ST_SB

Este estado executa a instrução SB (Store Byte). O próximo estado inicia o cálculo do endereço efetivo na memória e habilita a seleção de byte para a escrita. O próximo estado executa a escrita do byte na memória de dados, finalizando a instrução.

Estado: ST_SW

Este estado executa a instrução SW (Store Word). O próximo estado inicia o cálculo do endereço efetivo na memória e seleciona a fonte de dados para a escrita da palavra. O próximo estado executa a escrita da palavra na memória de dados, finalizando a instrução.

Estado: ST_JR

Este estado executa a instrução JR (Jump Register). O próximo estado inicia o cálculo do endereço de salto pela ULA, baseado no conteúdo de um registrador. O próximo estado atualiza o PC com o endereço de salto calculado, efetivando o desvio, e finaliza a instrução.

Estado: ST_J

Este estado executa a instrução J (Jump). O próximo estado indica o momento de atualizar o PC diretamente com o endereço de salto especificado na instrução, finalizando o desvio incondicional.

Estado: ST_JAL

Este estado executa a instrução JAL (Jump And Link). O próximo estado gerencia a escrita do endereço de retorno no registrador `$ra` e a atualização do PC com o endereço de salto, finalizando a instrução.

Estado: ST_SLL_SRA

Este estado executa as instruções SLL (Shift Left Logical) e SRA (Shift Right Arithmetic). O próximo estado configura a função de deslocamento e o registrador de destino para a operação. O próximo estado finaliza a escrita do resultado do deslocamento no registrador de destino. O estado “intermediário” garante que o contador continue avançando nos ciclos intermediários.

Estado: ST_BEQ

Este estado executa a instrução BEQ (Branch if Equal). O próximo estado inicia a comparação de igualdade na ULA e, se verdadeira, atualiza o PC condicionalmente com o endereço do desvio, finalizando a instrução.

Estado: ST_BNE

Este estado executa a instrução BNE (Branch if Not Equal). O próximo estado inicia a comparação de diferença na ULA e, se verdadeira, atualiza o PC condicionalmente com o endereço do desvio, finalizando a instrução.

Estado: ST_SLT

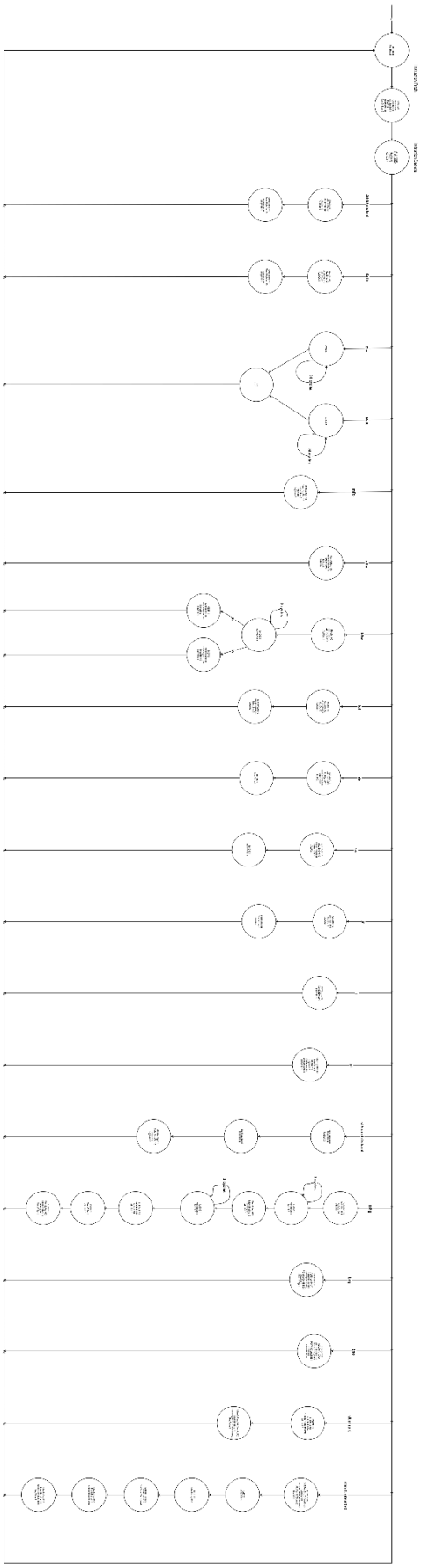
Este estado executa a instrução SLT (Set Less Than). O próximo estado configura a ULA para a operação "set if less than" e habilita a saída da ULA. O próximo estado finaliza a escrita do resultado (1 ou 0) no registrador de destino, após a comparação.

Estado: ST_SLLM

Este estado executa a instrução SLLM (Shift Left Logical by Memory). O próximo estado inicia o cálculo de um endereço para a leitura do valor de deslocamento. O próximo estado seleciona a memória de dados para a leitura do valor de deslocamento. O próximo estado captura o valor de deslocamento lido no MDR1. O próximo estado configura a unidade de deslocamento com o valor lido da memória. O próximo estado continua a configuração da unidade de deslocamento. O próximo estado finaliza a escrita do resultado do deslocamento no registrador de destino.

Estado: ST_XCHG

Este estado executa a instrução XCHG (Exchange). O próximo estado inicia a primeira parte da operação de troca, configurando a ULA para um cálculo inicial. O próximo estado lê o primeiro operando da memória, capturando-o no MDR1. O próximo estado prepara a ULA para um novo cálculo relacionado ao segundo operando. O próximo estado lê o segundo operando da memória, capturando-o no MDR2. O próximo estado prepara a ULA para a operação final de troca. O próximo estado inicia a escrita do primeiro operando de volta na memória. O próximo estado finaliza a escrita do segundo operando na memória, completando a troca, e reseta o processador. O estado “intermediário” garante que o contador continue avançando nos ciclos intermediários.



Acessível pelo link: [Descrição dos Estados do Controle]

Conjunto de Simulações

Não foi possível ser testado os módulos individualmente nem mesmo o conjunto de toda a cpu por questão de tempo, todavia, o projeto está em condições de ser posto em teste para que seja verificadas as simulações do funcionamento do processador se tudo ocorrer como desejado.

Conclusão

Este relatório detalhou o projeto e a implementação de uma Unidade Central de Processamento (CPU) desenvolvida para a disciplina, abordando desde a arquitetura geral até as especificidades de cada módulo e os estados de controle. Foram apresentadas as unidades funcionais essenciais, como a Unidade de Controle, a ULA, o Banco de Registradores e a Memória, e suas interconexões.

A "Descrição das Entidades" aprofundou nas funcionalidades de cada componente, como o Extensor de Sinal, o Byte Selector e as unidades de deslocamento, destacando seus objetivos e algoritmos. A "Descrição dos Estados do Controle" elucidou o funcionamento da máquina de estados finitos da Unidade de Controle, explicando as transições entre os estados de busca, decodificação e execução de diversas instruções, além de lidar com exceções e reset.

Embora o "Conjunto de Simulações" não tenha sido possível confecionar por questões de tempo, impedindo testes completos da CPU e de todos os módulos individualmente, o projeto atingiu um estágio de desenvolvimento que permite a verificação de sua funcionalidade.

Em suma, o trabalho representa um esforço significativo na concepção de uma CPU, fornecendo uma base sólida para futuras implementações e refinamentos. A arquitetura modular e a descrição detalhada dos componentes e do controle estabelecem um ponto de partida para a validação completa do processador em um ambiente de simulação mais abrangente.