**INSTITUTO FEDERAL DO ESPÍRITO SANTO**

**ENGENHARIA DE CONTROLE E AUTOMAÇÃO**

BRUNO LORENZO THOMAZ MARQUES

MARIANA PINTO FERRAZ TASSAN

**IMPLEMENTAÇÃO EM HARDWARE DE INSTRUÇÃO DE**

**PROCESSADOR ARM MONOCÍCLO**

Serra

2023

BRUNO LORENZO THOMAZ MARQUES

MARIANA PINTO FERRAZ TASSAN

**IMPLEMENTAÇÃO EM HARDWARE DE INSTRUÇÃO DE**

**PROCESSADOR ARM MONOCÍCLO**

Trabalho desenvolvido para a disciplina de Arquitetura de Computadores e Sistemas Embarcados do Curso de Engenharia de Controle e Automação, do Instituto Federal do Espírito Santo Campus Serra, como parte dos requisitos para conclusão da disciplina.

Prof. Rafael Emerick Zape de Oliveira

Serra

2023

**RESUMO**

O objetivo principal deste trabalho consiste na expansão das instruções de um processador ARM de ciclo único, bem como a compreensão da sua implementação, utilizando a linguagem de descrição de hardware SystemVerilog. Para realizar a simulação desse processador, foi utilizado o software ModelSim, desenvolvido pela Mentor Graphics.

Palavras-chave: ARM, Processador, SystemVerilog, Single-Cycle, ModelSim, Arquitetura de Computadores.

**1. INTRODUÇÃO**

Inicialmente foi feito um estudo aprofundado pelos integrantes do grupo para a familiarização com o código em SystemVerilog, entendendo seus respectivos módulos e ligações apresentadas em aula. O ambiente de simulação precisou ser instalado e testado para que a fase de desenvolvimento e implementação do projeto fosse iniciada.

**1.1 Processador ARM Single-Cycle**

O processador ARM Single-Cycle é composto pelo módulo de nível superior top e seus módulos internos correspondentes ao processador e às memórias de instrução e de dados: arm, imem e dmem, respectivamente.

Diagrama, Esquemático

Descrição gerada automaticamente A seguir, observa-se o esquemático do processor ARM Single Cycle:

Figura - Processador Single-Cycle completo

**2. TESTANDO O PROCESSADOR ARM MONOCICLO**

O primeiro passo foi a representação do esquemático correspondente ao HDL em SystemVerilog do processador ARM (figura 2).

Diagrama, Esquemático

Descrição gerada automaticamente

Diagrama, Esquemático

Descrição gerada automaticamente

**Diagrama, Esquemático

Descrição gerada automaticamente**

**Figura 2 - Esquemático do Processador ARM**

**2.1 Simulação do Processador ARM Single-Cycle**

O primeiro passo para a simulação do processador foi a leitura do arquivo memfile.s que contém as instruções em linguagem assembly. Dessa forma, foi possível prever o que acontece em cada ciclo de clock e preencher a tabela (figura 3).

Tabela

Descrição gerada automaticamente

**Figura 3 - Tabela do Single Cycle ARM**

A instrução final do STR escreverá o valor 0x00000007 no endereço 0x00000064.

Em seguida, o arquivo arm\_single.sv foi simulado por meio do software ModelSim da Mentor Graphics adicionando todos os sinais da tabela (figura 3) à janela. Após a execução, a forma de onda foi gerada (figura 4).

Uma imagem contendo Gráfico

Descrição gerada automaticamente

**Figura 4 –** **Forma de onda memfile.dat**

Essa simulação serviu para comprovar que o processador analisado está funcionando da maneira correta. Também foi possível validar que o resultado gerado está dentro do previsto (figura 3).

**3. MODIFICANDO O PROCESSADOR ARM MONOCICLO**

Para expandir as funções executadas pelo processador ARM, foi necessário entender como o código em SystemVerilog estava estruturado e como essas funções poderiam ser implementadas. As novas funções são: MOV, CMP, TST, EOR, LDRB, STRB e BL.

**3.1 Instrução MOV**

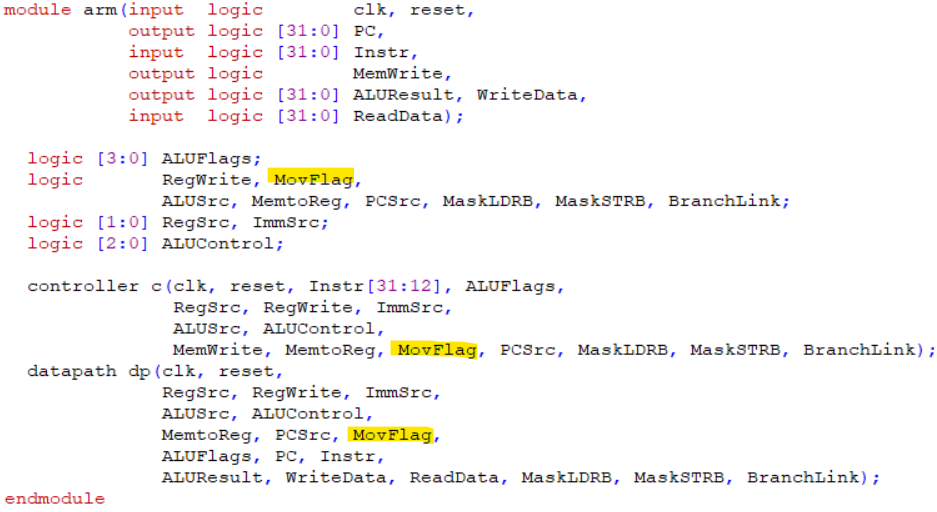
A instrução MOV é usada na arquitetura ARM para copiar ou mover dados de um local para outro. Ela pode atribuir um valor a um registrador, copiar um valor de um registrador para outro, carregar um valor imediato em um registrador ou armazenar um valor de um registrador na memória.

O primeiro passo para a implantação do MOV foi a habilitação das flags e extensão do decoder da ALU. Além disso, a flag NoWrite foi adicionada para indicar as instruções de processamento de dados que não escreverão na memória:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Os fios das novas flags foram ligados aos módulos do plano de controle e do plano de dados e aos seus respectivos módulos internos:



Em seguida, o fio auxiliar MovORAluResult é criado para receber a saída do novo mux criado para a instrução:



**3.2 Instrução CMP**

A instrução CMP é utilizada para comparar dois valores ao subtrair um do outro e atualizar as flags do processador com base no resultado da subtração. Essa instrução não guarda o valor da subtração.

Inicialmente, foi realizada a extensão do decoder da ALU com o novo comando para a instrução e definida a flag NoWrite para 1, de modo a indicar que o valor da operação realizada na ALU não será armazenado:

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

A flag NoWrite foi definida no plano de controle e seus fios foram ligados nos módulos decoder e condlogic. Por fim, a condição de escrita nos registradores exige que NoWrite seja 0:

Texto

Descrição gerada automaticamente

**3.3 Instrução TST**

A instrução TST realiza uma operação lógica AND bit a bit entre dois valores, mas não armazena o resultado da operação. Em vez disso, as flags do processador são atualizadas com base no resultado da operação lógica. Essa instrução é útil para realizar testes condicionais em instruções subsequentes do código.

As únicas alterações no código foram a expansão do decoder da ALU com o comando da instrução e a definição da ALUControl com a mesma decodificação da operação AND, salva a diferença na flag NoWrite, que para a instrução TST é definida como valor lógico alto:



**3.4 Instrução EOR**

A instrução EOR executa a operação lógica de OU exclusivo bit a bit entre os conteúdos dos registradores Rn e Rm e armazena o resultado no registrador de destino Rd. O conteúdo dos registradores Rn e Rm não é alterado durante a operação.

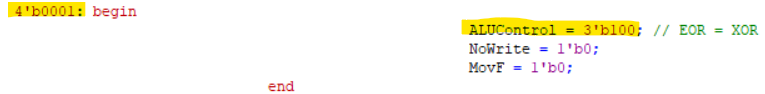
Essa instrução é frequentemente utilizada em operações de manipulação de bits, criptografia, cálculos de paridade e na implementação de algoritmos específicos.

Para a implementação da função EOR foi necessário aumentar o número de bits de ALUControl, localizada dentro do módulo da ALU, para acrescentar uma nova possibilidade de operação:

Texto

Descrição gerada automaticamente

Por fim, houve a expansão do decoder da ALU com o novo comando da instrução:



**3.5 Instrução LDRB**

LDRB significa "Load Register Byte" e é uma instrução de carga que transfere um valor de 8 bits (um byte) de um endereço de memória especificado para um registrador do processador.

A diferença da codificação entre as instruções LDR e LDRB está no segundo bit da função, ou seja, B (Byte). Como no código disponível já possui a instrução LDR implementada, para a implementação do LDRB, criou-se no decoder principal uma condição para saber se Byte (B) é 1 ou não, pois de acordo com o tipo de operação, LDRB possui esse bit com valor 1. Também já foi deixada a condição pronta para a implementação do STRB:

Linha do tempo

Descrição gerada automaticamente

Uma flag de controle MaskLDRB foi criada para identificar quando é LDRB ou LDR. Assim, foi preciso aumentar controls de 10 bits para 11, e futuramente aumentado para 13 bits para implementar (BL e STRB). Em seguida, foi feita a ligação do fio MaskLDRB nos módulos de plano de dados e de controle:

Texto

Descrição gerada automaticamente com confiança média

Por fim, foi feito a criação da lógica para uma máscara de bits no datapath, para aplicar no ReadData, criando uma variável auxiliar rd e substituindo ReadData por rd no multiplexador resmux:

Linha do tempo

Descrição gerada automaticamente com confiança média



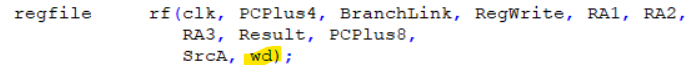
**3.6 Instrução STRB**

A instrução STRB (Store Register Byte) é utilizada para armazenar um byte de um registrador na memória. Essa instrução realiza a operação inversa da instrução LDRB.

Assim como em LDRB, a diferenciação entre as funções STR e STRB está no segundo bit da função, B (Byte). Portanto, a condição aplicada anteriormente também serve para diferenciar STR de STRB.

Criou-se uma flag de controle MaskSTRB e foram feitas as ligações dos fios nos módulos. Além disso, o controle teve seu número de bits aumentado, conforme já explicado na implementação da instrução anterior.

Após isso, foi feito a criação da lógica para máscara de bits no plano de controle para aplicar no WriteData, criando uma variável auxiliar wd e substituindo WriteData por wd no regfile:



Texto

Descrição gerada automaticamente

**3.7 Instrução BL**

A instrução de branch linkado (BL) é utilizada para realizar um salto para um endereço de destino específico, enquanto simultaneamente salva o endereço de retorno no registrador de ligação (Link Register, LR). O endereço de retorno é o endereço da instrução seguinte à instrução BL.

Inicialmente, criou-se uma nova flag BLFlag para identificar quando seria Branch ou Branch Linkado. Após as criações das flags para LDRB, STRB e BL, controls ficou com 13 bits. Uma condição também foi definida para identificar quando a instrução de Branch seria linkada (L = 1) ou não (L = 0).

Em seguida, um novo multiplexador foi criado no caminho de dados (datapath) para definir quando o número do registrador utilizado for 14, que representa LR, para o caso da flag BranchLink for 1, ou um registrador de destino qualquer quando BranchLink for 0. Em seguida, o resultado do mux armazenado em RA3 é enviado ao módulo regfile, bem como o endereço da próxima instrução (PCPlus4):

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

No módulo regfile, surge uma variável auxiliar WriteDataPC de 32 bits que é utilizada para receber a próxima instrução a ser escrita na memória caso BranchLink seja 1, ou a instrução atual caso BranchLink seja 0. Por último, caso RegWrite seja 1, o registrador 14 recebe a próxima instrução:

Texto

Descrição gerada automaticamente

**4.0 Repositório GIT e vídeo YouTube**

GIT: <https://github.com/brunolorenzo/ARMProcessors>

Youtube: <https://www.youtube.com/watch?v=batwG1hlkNE>