Trabalho Final – Organização e Arquitetura de Computadores
Projeto RISC-V Multiciclo

Aluno(a) 1: Gustavo Pereira Chaves
Aluno(a) 2: Júlia Yuri Garcia Baba
Matrícula 1: 19/0057921
Matrícula 2: 19/0014113

1. Objetivos

Montar e simular uma versão do processador RISC-V multiciclo, em VHDL, utilizando de base os trabalhos desenvolvidos ao longo da disciplina, interligando os módulos relativos à parte operativa e à parte de controle.

Ademais, também deve-se implementar as instruções adicionais: ori, xori e andi.

2. Descrição do trabalho

- **Descrição da implementação**

Primeiramente, de modo sucinto, para iniciar a simulação do um RISC-V multiciclo, temos cinco estados iniciais:

1. ***Busca da instrução***: Libera a escrita de PC, para que este possa ser incrementado, prepara a ULA para soma e seleciona PC através dos multiplexadores, porém antes salva o valor antigo de pc em pcback. Também, salva a instrução vinda da memória, ativando a escrita do registrador de instruções.
2. ***Decodificação e leitura dos registradores***: Libera a leitura do banco de registradores por meio do seu seletor e carrega o rs1 e rs2 nos registradores A e B. Verifica o opcode e funcs para determinar qual estado tratar, além disso já aproveita a ULA para calcular o desvio das instruções branch ou jal., usando o pcback salvo anteriormente.
3. ***Cálculo de endereço***: Depende da instrução:
   i. **Tipo R**: são realizada as operações desse tipo na ULA. O controle usa os campos funct3 e funct7, além da ULAop provido do controle principal, para decidir qual operação será realizada, logo após, armazena o resultado no registrador de saída da ULA.
   ii. **Instruções lw e sw**: é utilizado a ULA para calcular o endereço de memória a ser lido ou escrito, logo depois, rmazena o resultado no registrador de saída da ULA.
   iii. **Instrução beq**: a ULA deve comparar se os registradores da instrução beq são iguais, através de uma subtração. Caso isso se confirme, PC recebe o endereço de desvio.
   iv. **Instrução jal**: o registrador PC recebe o valor de PC+4, calculado na primeira etapa, liberando o contador de programas. Depois, PC recebe o endereço de desvio, calculado na segunda etapa e armazenado no registrador da ULA. Note que a ULA não é utilizada nessa etapa.
4. ***Acesso a memória e conclusão do tipo R***:
   Novamente, depende da instrução:
   i. **Instrução sw**: armazena o conteúdo de um registrador na memória de endereço, assim como, armazena no registrador de saída da ULA
   ii. **Instrução lw**: lê o conteúdo da memória de endereço e armazena no registrador de dados
5. ***Conclusão lw***: Armazena o valor lido da memória, que foi colocado no registrador de dados na etapa anterior, no registrador de destino.

Etapa 1	Reg – inst <= memória[PC] Pcback <= PC(anterior) PC <= PC + 4;	
Etapa 2	Reg A <= Reg[reg inst[19:25]] Reg A <= Reg[reg inst[24:20]] Reg ULA <= Pcback + (imm «1)	
Etapa 3	Tipo R	Reg ULA <= Reg A op Reg B
	LW/SW	Reg ULA <= Reg A + imm
	BEQ	1. Resultado ULA <= Reg A - Reg 2. se saída zero da ULA = 1 3. PC <= Reg ULA
	JAL	1. Reg[rd] <= PC+4 2. PC <= Reg ULA
Etapa 4	Tipo R	Reg[11:7] <= Reg ULA
	SW	mem[reg ULA] <= Reg B
	LW	Reg dados <= mem[reg ULA]
	AUIPC	
Etapa 5	Reg[11:7] <= Reg dados	

Fig.1. Estados e suas devidas operações

É importante salientar, que para que os estados iniciais pudessem ser implementados, foi utilizado as seguintes entradas: clock (1 bit), instrução (32 bits), opcode (7 bits), func7 (7bits), func3 (3 bits).

- **Instruções implementadas**

Já para a execução das instruções, deve-se ter em mente o seu tipo:

1. ***Tipo R:*** Tem o funcionamento básico de todas suas instruções pautadas em selecionar a saída dos multiplexadores referentes às entradas dos registradores, em seguida, ativa a operação da ULA. Feito isso, o registrador de destino recebe o resultado da operação aritmética.
   a) ***add***: Soma rs1 com rs2 e salva o resultado em rd.
   b) ***sub***: Realiza a subtração entre rs1 e rs2 e salva o resultado em rd.
   c) ***and***: Realiza a operação lógica AND bit a bit entre rs1 e rs2 e salva o resultado em rd.
   d) ***nand:*** Como NAND é uma instrução emulada, quando o programa é convertido no RARS, essa se transforma em duas instruções, AND e NOT. Sendo assim, ambas as operações serão realizadas na ULA e depois o resultado será salvo em rd.
   e) ***or:*** Realiza a operação lógica OR bit a bit entre rs1 e rs2 e salva o resultado em rd.
   f) ***nor:*** Como NOR é uma instrução emulada, quando o programa é convertido no RARS, essa se transforma em duas instruções, OR e NOT. Sendo assim, ambas as operações serão realizadas na ULA e depois o resultado será salvo em rd.
   g) ***xor:***Realiza a operação lógica XOR bit a bit entre rs1 e rs2 e salva o resultado em rd.
   h) ***slt:*** Compara os valores de rs1 e rs2. Caso, a ULA determine que rs1 < rs2, rd deverá receber 1 (em 32 bits) e 0 caso contrário.
2. ***Tipo I***
   a) ***lw:*** rd recebe, o valor guardado na posição de memória, carregado a partir da memória de dados, representada pelo resultado da soma aritmética, vindo da ULA, entre rs1 e a constante dada (imediato),
   b) ***addi***: rd recebe o resultado da soma aritmética, vindo da ULA, entre o valor de rs1 e o imediato de sinal extendido.
   c) ***ori***: rd recebe o resultado da operação lógica OR-inclusivo bit a bit, vindo da ULA, entre o valor de rs1 e o imediato de sinal extendido.
   d) ***andi***: rd recebe o resultado da operação lógica AND bit a bit, vindo da ULA, entre o valor de rs1 e o imediato de sinal extendido.
   e) ***xori***: rd recebe o resultado da operação lógica XOR bit a bit, vindo da ULA, entre o valor de rs1 e o imediato de sinal extendido.
   f) ***jalr***: rd recebe o valor atual de PC e o PC recebe o resultado da soma

aritmética, vindo da ULA, entre o valor de rs1 e a constante dada (imediato).

3. ***Tipo S***
   a) ***sw:*** rs2 recebe os 12 primeiros bits da posição de memória representada pela soma aritmética, vinda da ULA, entre o valor de rs1 e o imediato.
4. ***Tipo SB***
   a) ***beq:*** PC é deslocado de acordo com o imediato, caso a ULA, determine que os valores de rs1 e rs2 são iguais.
   b) ***bne:*** PC é deslocado de acordo com o imediato, caso a ULA, determine que os valores de rs1 e rs2 são diferentes.
5. ***Tipo U***
   a) ***lui:*** rd recebe, entre os bits 31 e 11, o valor do imediato.
   b) ***auipc:*** Adiciona o valor imediato com sinal estendido, deslocada para a esquerda, ao pc, e escreve o resultado em rd.
6. ***Tipo UJ***
   a) ***jal:*** rd recebe o valor atual de PC e o PC recebe o resultado da soma aritmética, vindo da ULA, entre o antigo valor de PC e o imediato.

É importante notar que, toda vez que se menciona os registradores, seus valores são sempre carregados a partir do banco de registradores, assim como, a ULA é sempre ativada por meio de sinais vindo da unidade de controle. Além disso, antes de qualquer operação realizada na ULA, a saída dos multiplexadores é selecionada com relação a instrução a ser executada, podendo essas saídas serem rs1, rs2, imm e pc.

Outro ponto, é a pseudo-instrução NOT, como explicado no trabalho de geração de dados imediatos, essa é estruturada através da operação lógica XORI (XORI rd, rs1, -1), ou seja, causa uma inversão lógica bit a bit do registrador fonte.

- **Dificuldades encontradas**

Uma das primeiras dificuldades encontradas foi o carregamento da memória. Como pode-se observar, no diagrama do circuito, a memória já estava carregada, contudo, no programa precisamos testá-la através da leitura de um arquivo. Para esse problema criou-se um estado prévio ao de busca, chamado de estado zero. Dessa forma, foi feita uma função de pré-processamento, que se encarrega de fazer tal ação, de forma similar ao que foi desenvolvido no testbench do projeto de Memória de Dados e Instruções.

Já no estado de busca, o obstáculo encontrado foi a incrementação do pc sem alterar o pcback e o reg de instrução. Sendo assim, para contornar tal situação, utilizou-se um reg interno para armazenar o valor e apenas repassar este quando o próximo estado for ativado.

3. Testes realizados

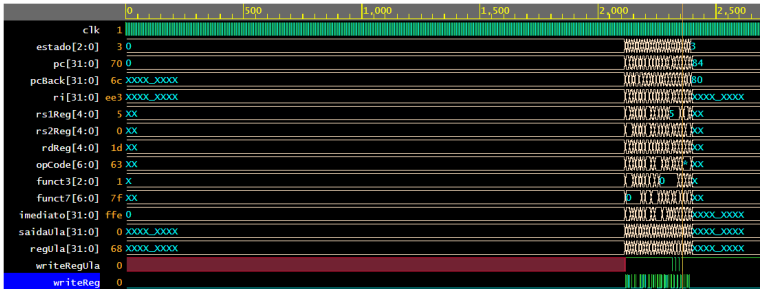

Fig.2. Visão geral das formas de onda (0 ns até 2500 ns)

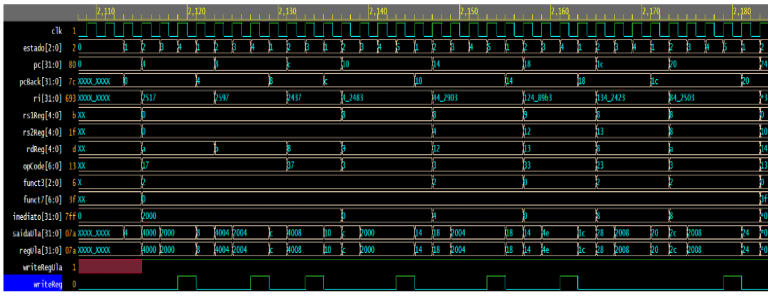

Fig.3. Forma de onda (2110 ns até 2180 ns)

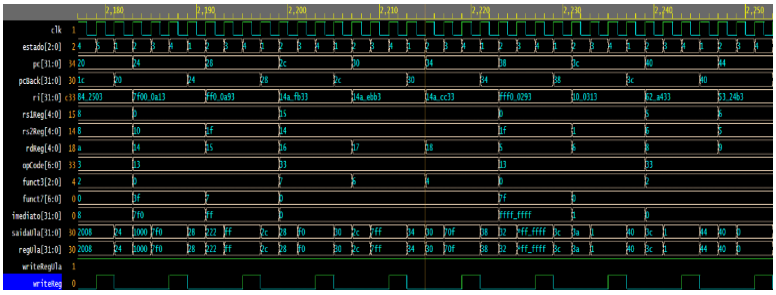

Fig.4. Forma de onda (2180 ns até 2250 ns)

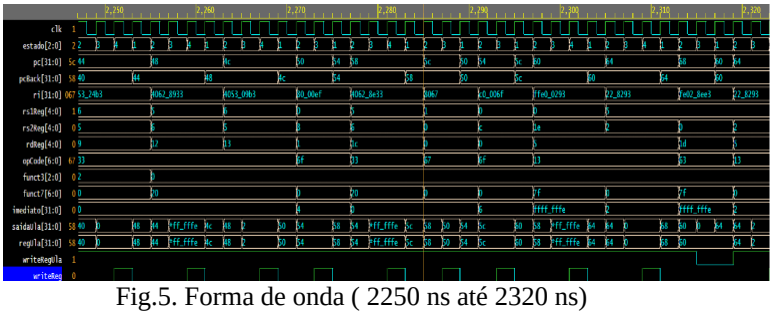

Fig.5. Forma de onda (2250 ns até 2320 ns)

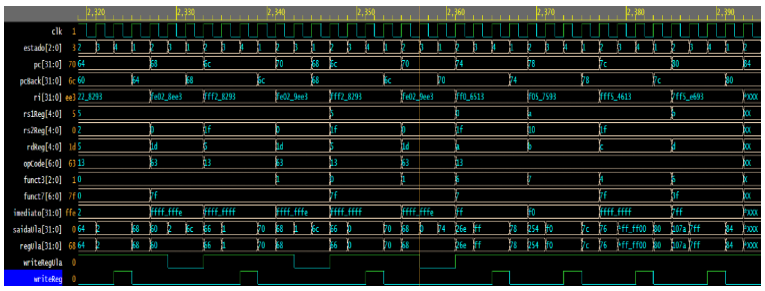

Fig.6. Forma de onda (2320 ns até 2390 ns)

Para os testes foi utilizado o arquivo asm passado pelo professor, exportando as instruções e os dados, pelo RARS, em forma de um arquivo txt hexadecimal. Por conta disso, na fig.2 vemos um atraso até a inicialização do programa, que se deve a fase de carregamento.

Através das formas de onda foi possível analisar o comportamento da arquitetura desenvolvida, tendo em vista o que era esperado pelo RARS.

4. Conclusão

Portanto, com a realização dos testes pode-se se concluir que todas as instruções funcionam como esperado, demonstrando que os conhecimentos aprendidos durante o semestre puderam ser executados na prática.

Ademais, com relação as dificuldades relatadas, é possível concluir, que apesar de uma prévia familiaridade com a linguagem, o fato dela não ser sequencial junto com o tamanho do circuito, fez com que ambas tivessem base nisso.