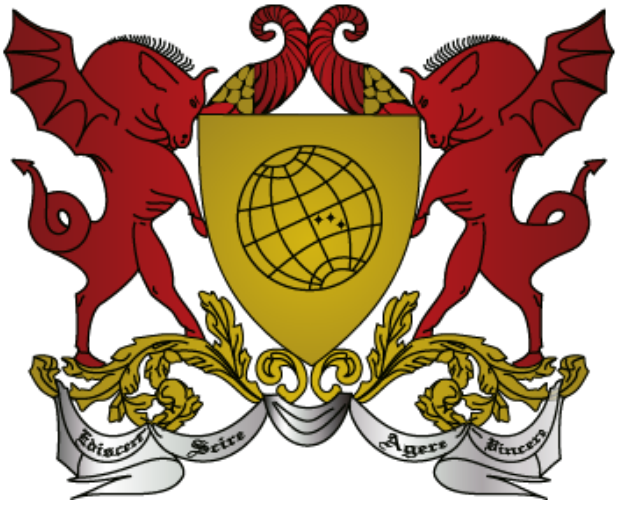

Universidade Federal de Viçosa - Campus Florestal

Ciência da Computação - 3º Período

Trabalho Prático de OC1 - 02

Caminho de dados do MIPS

Bruno Marra - 3029
Gustavo Viegas - 3026

Florestal

2017

1. Introdução

O trabalho prático foi divido em duas partes. A primeira parte consistia em, a partir de um caminho de dados fornecido na especificação, que se fosse feito a implementação do mesmo, utilizando a linguagem Verilog HDL, após isso também era necessário que fosse gerado o diagrama de ondas para o acompanhamento do caminho desenvolvido.

Para a segunda parte do trabalho, após as alterações necessárias no nosso código escrito em Verilog, usamos uma placa FPGA, onde fizemos a síntese do código e rodamos o mesmo através do hardware solicitado.

2. Metodologia

O caminho de dados fornecido na especificação foi a base utilizada para que pudéssemos elaborar nosso código, a Figura 1 mostra esse caminho com os dados necessários.

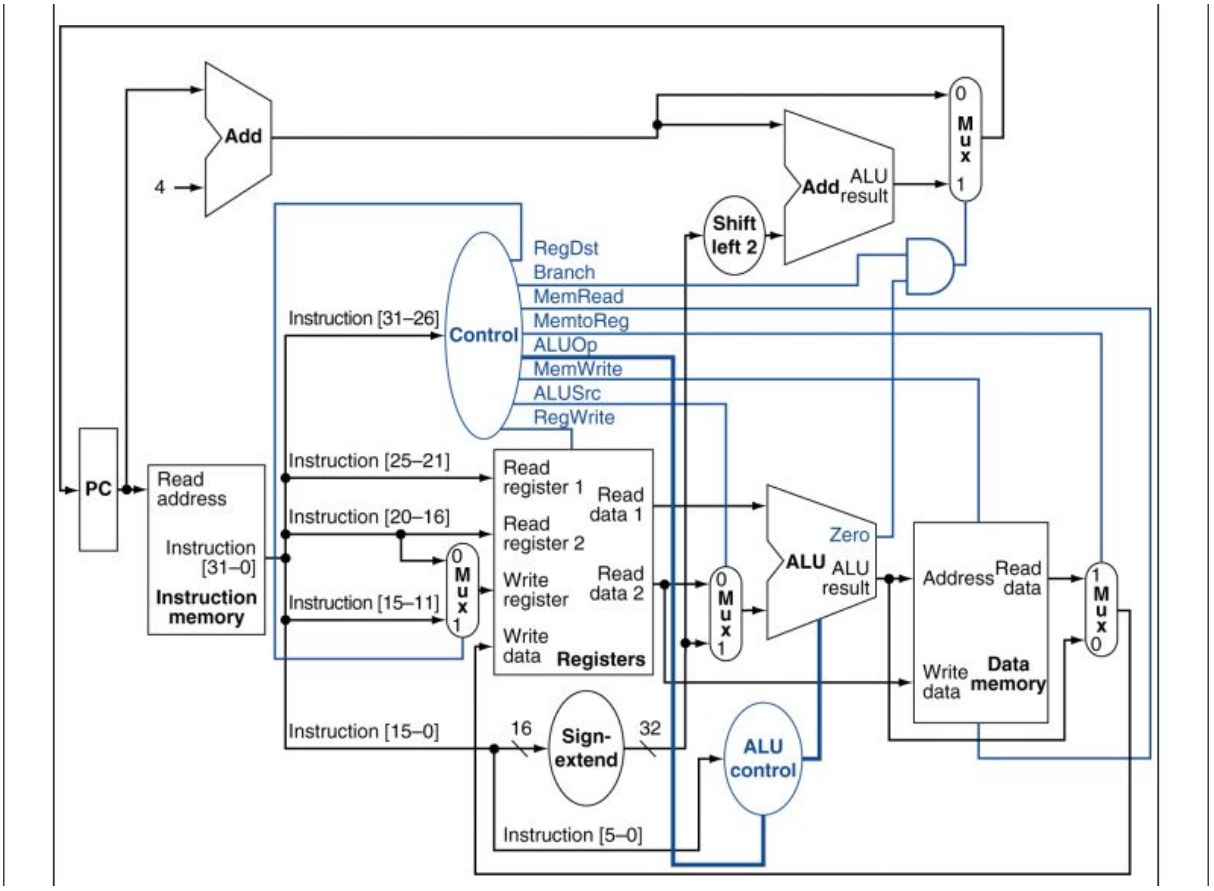

Figura 1 - Caminho de dados do MIPS

Tendo o caminho de dados em mãos, notamos que cada parte desse caminho deveria ser um módulo, então adotamos um arquivo .v para cada módulo e separamos devidamente o código, tanto para facilitar na identificação quanto para termos um entendimento melhor do processo como um todo. O primeiro módulo que tratamos foi o PC[1] (Program Counter), que consiste num registrador que armazena sempre o endereço da próxima instrução. A Figura 2 exemplifica esse módulo criado em Verilog.

```
1   module pc (
2     input wire [31:0] nextAddress,
3     input clock,
4     output reg [31:0] readAddress
5   );
6     initial begin
7       readAddress <= 0;
8     end
9
10    always @(posedge clock) begin
11      readAddress <= nextAddress;
12    end
13  endmodule
```

Figura 2 - Módulo PC

A partir disso, criamos então o próximo módulo, o Instruction Memory, que é responsável por guardar as instruções disponíveis para execução. o Módulo criado é exemplificado na Figura 3.

```
1   module instructionMemory (
2     input wire reset,
3     input wire [31:0] readAddress,
4     output reg [31:0] instruction
5   );
6
7     reg [31:0] memory [0:5]; // 6 instrucoes de 32 bits
8
9     integer i;
10
11    initial begin
12      $readmemb("instructions.bin", memory);
13      $display("rdata:");
14      for (i = 0; i < 6; i= i + 1)
15        $display("%d: %b", i, memory[i]);
16    end
17
18    always @(readAddress) begin
19      instruction <= memory[readAddress];
20      $display("# %t, RA: %d, %b", $time, readAddress, instruction);
21    end
22
23  endmodule
```

Figura 3 - Módulo instructionMemory

[1] Referência do PDF: https://www.utdallas.edu/~dodge/EE2310/lec13.pdf

Na leitura do arquivo binário, temos as instruções aceitas pelo programa, onde ele é consultado a cada ciclo de clock. Após lida a instrução, ele segue o caminho seguinte, passando por um MUX de duas entradas simples de 32 bits ou diretamente para o banco de registradores.

Assim que os registradores são coletados, eles são enviados para a ALU, uma ULA responsável por fazer a operação necessária sobre o dado enviado. O resultado que sai da ALU é armazenado na memória de dados, onde o próximo ciclo é acionado. Boa parte das operações tratadas neste TP segue esse caminho.

3. Resultados

Para a primeira parte do trabalho, tivemos o resultado a partir do diagrama de ondas gerado, onde pudemos acompanhar a variação dos wires a cada ciclo de clock, bem como o valor de cada módulo. A Figura 4 exemplifica alguns momentos do diagrama de ondas. O diagrama completo pode ser observado por meio do software GTKWave.

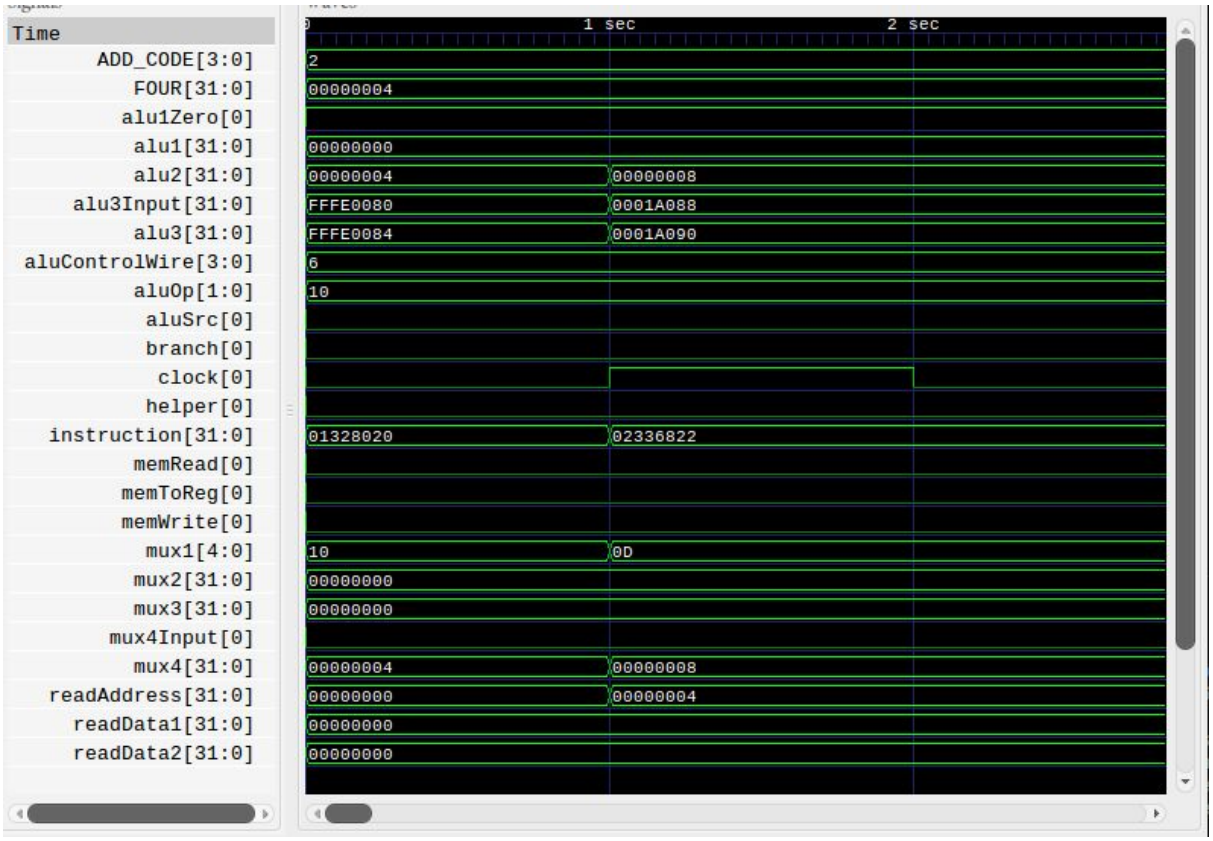

Figura 4 - Diagrama de ondas (GTKWave)

4. Conclusão

O trabalho mostra o grau de complexidade de uma máquina como o MIPS, visto que foi feita uma implementação relativamente simplificada da máquina, sem o uso de Pipelines ou instruções mais complexas, como Jumps, etc. Logo, podemos observar também na prática como se é dado o caminho de dados efetivamente, resultando numa melhor absorção do conteúdo aprendido em sala, bem como as observações e a prática da linguagem Verilog HDL.

5. Referências

Tanenbaum, Andrew S. - **Organização Estruturada de Computadores - 5ª Ed**, Pearson - Addison Wesley; 2006.