

Universidade Federal de Viçosa - Campus Florestal

Ciência da Computação - 3º Período

## **Trabalho Prático de OC2 - 01**

Caminho de dados do MIPS com Pipeline

Bruno Marra - 3029  
Gustavo Viegas - 3026

**Florestal**

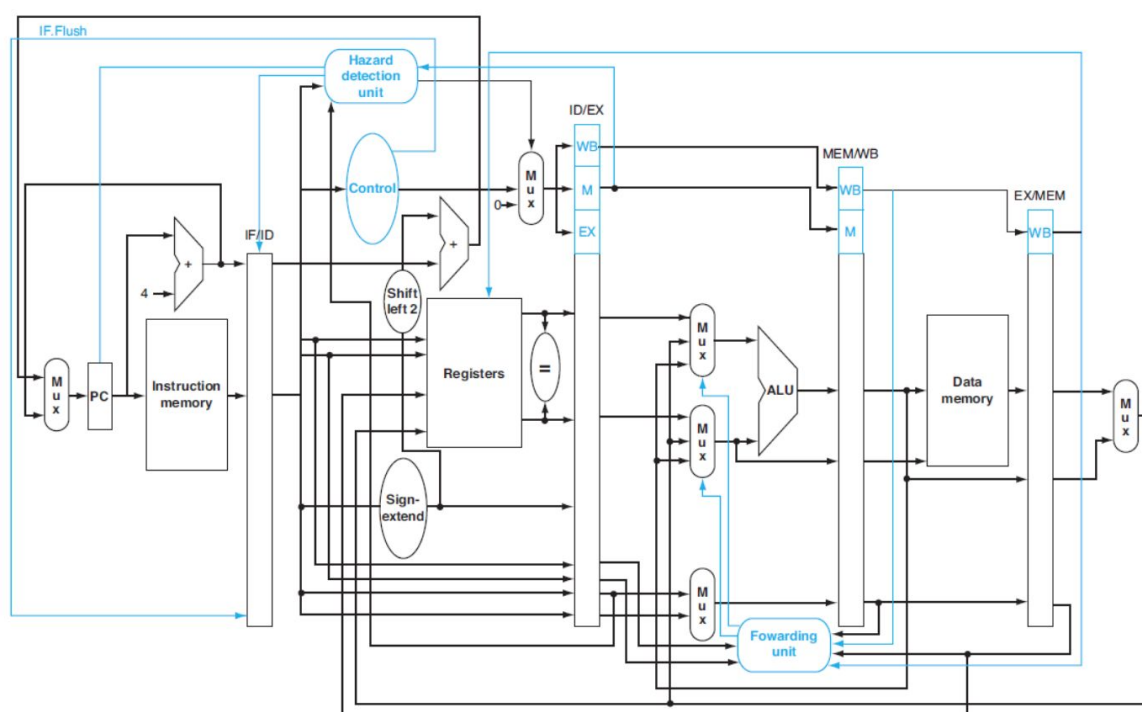
**2017**

## 1. Introdução

O trabalho prático foi dividido 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. O caminho de dados completo incluía a implementação da sequência de pipeline e também unidades de controle de detecção de hazards e forwardings.

## 2. Metodologia

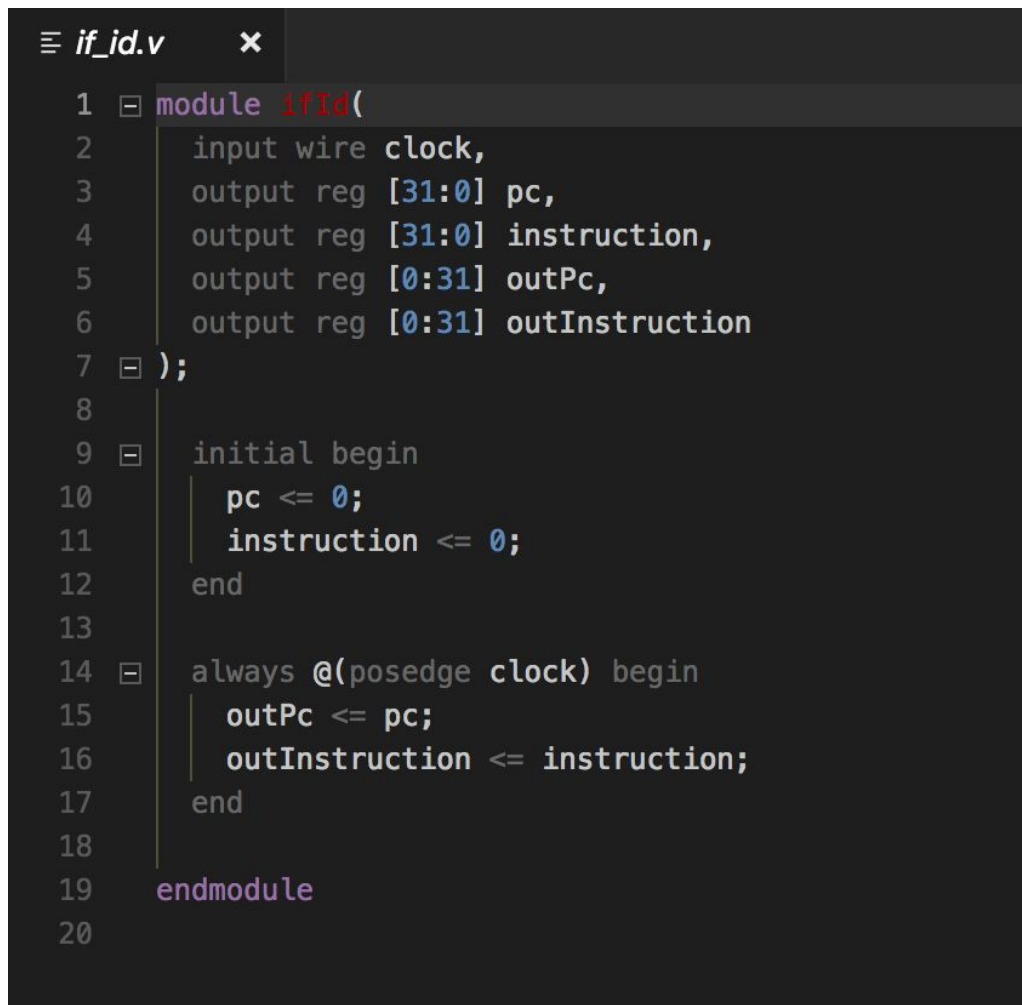
O caminho de dados fornecido na especificação foi a base utilizada para que pudéssemos elaborar nosso código, a Figura 1 mostra esse caminho com os dados necessários.



**Figura 1 - Caminho de dados do MIPS com pipeline**

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. A partir dos módulos já implementados no TP2 de OC1, fizemos as modificações necessárias e criamos primeiramente os módulos de pipeline, IF\_ID, ID\_EX, EX\_MEM, MEM\_WB, e usamos as especificações do livro para fazermos a ligação dos fios. Basicamente cada unidade do pipeline armazena os dados dos fios por um posedge clock, os inicializa com 0 e passa adiante o valor do fio enquanto armazena o da instrução seguinte. Um exemplo desses módulos está representado na Figura 2.



```
1 module ifId(
2     input wire clock,
3     output reg [31:0] pc,
4     output reg [31:0] instruction,
5     output reg [0:31] outPc,
6     output reg [0:31] outInstruction
7 );
8
9 initial begin
10     pc <= 0;
11     instruction <= 0;
12 end
13
14 always @(posedge clock) begin
15     outPc <= pc;
16     outInstruction <= instruction;
17 end
18
19 endmodule
20
```

**Figura 2 - Módulo IF\_ID**

Após implementarmos todas as camadas do pipeline, partimos para a implementação do módulo de detecção de forwarding, módulo este implementado de acordo com as instruções do livro, visto que não encontramos nenhum outro lugar de referência que pudesse nos auxiliar na execução do mesmo, no qual levamos muito tempo para implementar o que prejudicou a criação do módulo de detecção de hazards, visto que não conseguimos fazer o

módulo de detecção funcionar da forma como deveria. O código da implementação do módulo é apresentado na Figura 3.

```
1 module forwarding_v
2   input wire clock,
3
4   // INPUTS
5   input wire [4:0] rs,
6   input wire [4:0] rt,
7
8   // EX/MEM
9   input wire [4:0] writeRegister, // registerRD
10  input wire regWrite,
11
12  // MEM/WB
13  input wire [4:0] writeRegisterMemWb, // registerRD
14  input wire regWriteMemWb,
15
16  // OUTPUTS
17  output reg [1:0] fowardA,
18  output reg [1:0] fowardB
19  ;
20
21  initial begin
22    fowardA <= 0;
23    fowardB <= 0;
24  end
25
26  always @(posedge clock) begin
27
28    // Foward EX
29    if (regWrite && (writeRegister != 0) && (writeRegister == rs))
30      fowardA <= 2'b10;
31    else if (regWriteMemWb && (writeRegisterMemWb != 0) && !(regWrite && (writeRegister != 0)) && (writeRegister != rs) &&
32      fowardA <= 2'b01;
33    else
34      fowardA <= 2'b00;
35
36    // Foward MEM
37    if (regWrite && (writeRegister != 0) && (writeRegister == rt))
38      fowardB <= 2'b10;
39    else if (regWriteMemWb && (writeRegisterMemWb != 0) && !(regWrite && (writeRegister != 0)) && (writeRegister != rt) &&
40      fowardB <= 2'b01;
41    else
42      fowardB <= 2'b00;
43  end
44 end
45
46 endmodule
47
```

**Figura 3 - Módulo de detecção de forwarding**

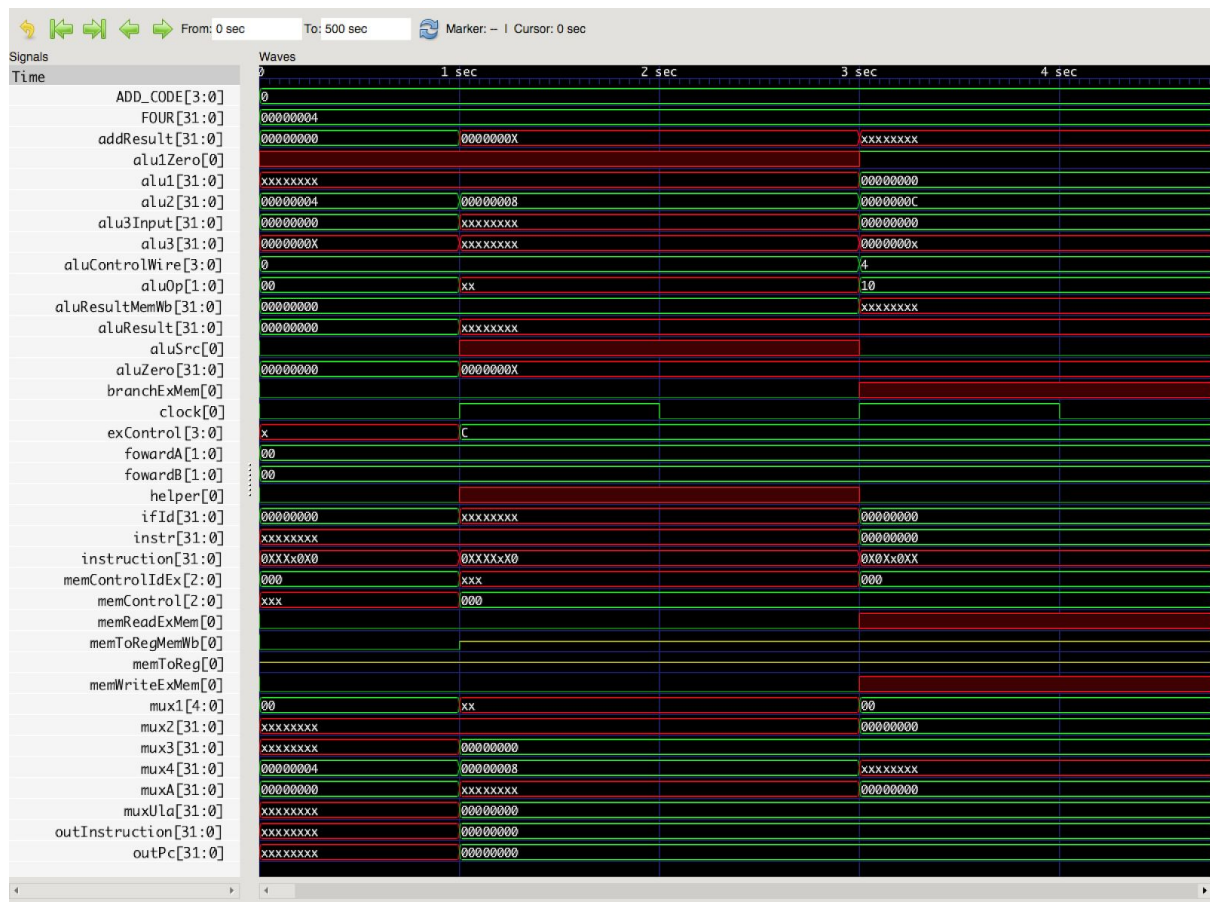
Na leitura do arquivo binário, temos as instruções aceitas pelo programa, onde ele é consultado a cada ciclo de clock. Após lida a instrução, ele segue o caminho seguinte, passando por um MUX de duas entradas simples de 32 bits ou diretamente para o banco de registradores.

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, porém com a implementação usando pipeline a complexidade aumentou severamente, mesmo sem 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.

Caso fosse finalizado o caminho funcionando totalmente, poderíamos observar que mesmo com a implementação do pipeline, os resultados de saída da ALU e de execução

seriam reduzidos, ou seja, seriam necessários menos ciclos de clock para executar cada instrução. Sendo assim, teríamos os mesmos resultados com um ganho significativo de desempenho dos mesmos.

## **5. Referências**

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