Trabalho 2 de Arquitetura de Computadores I
Turma 590 - 2020/2
Professor: Cesar Augusto Missio Marcon
Alunos: Diego Henrique Oliveira; Ícaro Stumpf

Projeto – - *Implementar a primeira versão do MIPS pipeline*

Descrição: O objetivo desta etapa é desenvolver uma arquitetura MIPS empregando pipeline de cinco estágios. Esta implementação deverá ser uma evolução da versão multiciclo, explorada na fase anterior e **não deverá dar suporte ao tratamento de conflitos de dados ou controle.**

A solução dos conflitos de dados deverá ser feita em software, a partir da inserção de ***nops*** entre instruções que tenham dependência verdadeira ou ainda através da reordenação das instruções. Solução similar deverá ser aplicada para conflito de controle, ou seja, inserindo instruções de ***nop*** logo após a instrução de salto para evitar que instruções incorretas sejam carregadas para o pipeline. É pressuposto que o ***testbench*** apresentado na primeira etapa do trabalho seja reutilizado aqui, com as inserções de ***nops*** necessárias para o funcionamento do mesmo.

Para o desenvolvimento do presente trabalho, foi utilizada, como base, a implementação em VHDL da MIPS multiciclo desenvolvida pelos professores Fernando Moraes e Ney Calazans, versão junho de 2020[1].

Essa primeira versão da MIPS Pipeline é, portanto, uma adaptação da MIPS muticiclo e foram feitas conforme orientações do professor César Marcon[2]

Em termos de instruções, apenas foram acrescentadas as linha de código necessárias à inserção da instrução NOP (que funciona como uma bolha, evitando conflitos).

As “barreiras temporais” (registradores entre os estágios do pipeline), bem como os sinais criados para propagação da informação entre elas ficarão evidentes ao se analisar os segmentos de código introduzidos no arquivo original, conforme se vê abaixo:

[1] Disponível em http://www.inf.pucrs.br/~calazans/undergrad/orgcomp_EC/mips_multi/Files_MIPS-MC_SingleEdge.7z, acesso em 30 de Setembro de 2020
[2] Disponível em https://youtu.be/LGMB6kDdI-g, acesso em 30 de Setembro de 2020

```
88  -- Pipeline Registers
89  --+++++++++++++++
90
91      type BIDI is record
92              npc : std_logic_vector(31 downto 0); -- pc + 4
93              IR : std_logic_vector(31 downto 0); -- instruction code
94      end record;
95
96      type DIEX is record
97              npc: std_logic_vector(31 downto 0); -- pc + 4
98              RA : std_logic_vector(31 downto 0); -- Rs loaded value
99              RB : std_logic_vector(31 downto 0); -- Rt loaded value
100             IMED : std_logic_vector(31 downto 0); -- Immediate value
101             RS : std_logic_vector(4 downto 0); -- Rs address
102             RD : std_logic_vector(4 downto 0); -- Rd address
103             RT : std_logic_vector(4 downto 0); -- Rt address
104             uins : microinstruction;
105         end record;
106
107     type EXMEM is record
108             npc : std_logic_vector(31 downto 0); -- pc + 4
109             salta : std_logic;
110             RALU : std_logic_vector(31 downto 0); -- ULA's output
111             adD : std_logic_vector(4 downto 0); -- Write Bank Address
112             RB : std_logic_vector(31 downto 0); -- Rt loaded value (for sw instructions)
113             uins : microinstruction;
114         end record;
115
116         type MEMER is record
117             npc : std_logic_vector(31 downto 0); -- pc + 4
118             adD : std_logic_vector(4 downto 0); -- Write Bank Address
119             RALU : std_logic_vector(31 downto 0); -- ULA's output
120             MDR : std_logic_vector(31 downto 0); -- read memory value
121             uins : microinstruction;
122         end record;
123
124 end p_MIPS_Pipeline;
```

Figura 1.Registradores do Pipeline

```
--=============================================================================
-- BI/DI Pipeline Register
--=============================================================================

process(ck, rst)
begin
    if rst = '1' then
        bidi.npc <= (others=>'0');
        bidi.IR <= (others=>'0');
    elsif ck'event and ck = '0' then
        bidi.npc <= incpc; -- store in the pipeline register pc + 4
        bidi.IR <= instruction; -- store in the pipeline register instruction code
    end if;
end process;
```

```
--=============================================================================
-- DI/EX Pipeline Register
--=============================================================================

process(ck, rst)
begin
    if rst = '1' then
        diex.npc <= (others=>'0');
        diex.RD <= (others=>'0');
        diex.RT <= (others=>'0');
        diex.RS <= (others=>'0');
        diex.RA <= (others=>'0');
        diex.RB <= (others=>'0');
        diex.IMED <= (others=>'0');
        -- uins rst
        diex.uins.inst_branch <= '0';
        diex.uins.inst_grupol <= '0';
        diex.uins.inst_grupoI <= '0';
    diex.uins.wreg <= '0';
    diex.uins.ce <= '0';
    diex.uins.rw <= '0';
    diex.uins.bw <= '1';
    diex.uins.i <= NOP;

    elsif ck'event and ck = '0' then
        diex.uins <= uins;
        diex.npc <= bidi.npc;
        diex.RD <= IR(15 downto 11);
        diex.RT <= IR(20 downto 16);
        diex.RS <= IR(25 downto 21);
        diex.RA <= R1;
        diex.RB <= R2;
        diex.IMED <= cte_im;
    end if;
end process;
```

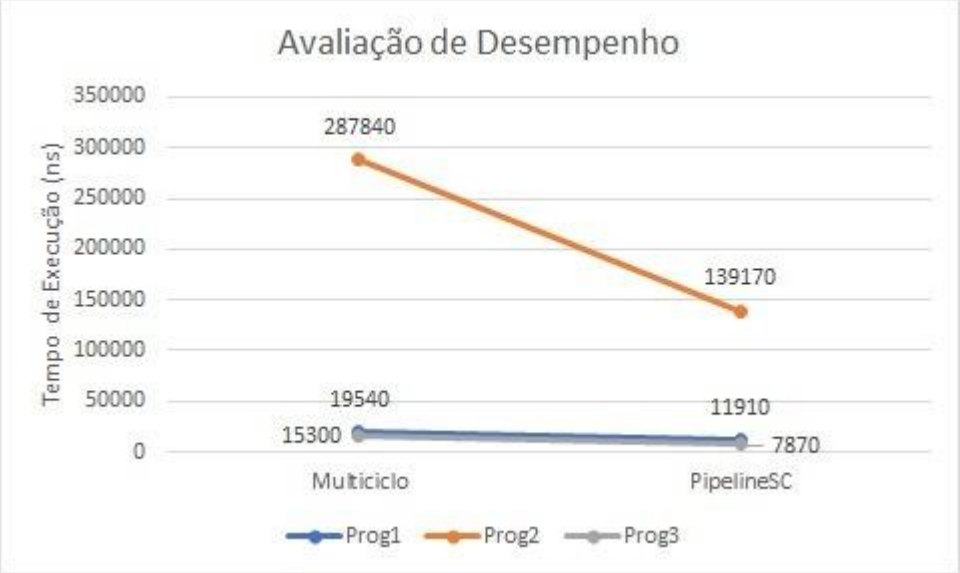
```
--=============================================================================
-- EX/MEM Pipeline Register
--=============================================================================

process(ck, rst)
begin
     if rst = '1' then
          exmem.npc <= (others=>'0');
          exmem.salta <= '0';
          exmem.RALU <= (others=>'0');
          exmem.adD <= (others=>'0');
          exmem.RB <= (others=>'0');
          -- uins rst
          exmem.uins.inst_branch <= '0';
          exmem.uins.inst_grupol <= '0';
          exmem.uins.inst_grupoI <= '0';
          exmem.uins.wreg <= '0';
          exmem.uins.ce <= '0';
          exmem.uins.rw <= '0';
          exmem.uins.bw <= '1';
          exmem.uins.i <= NOP;

     elsif ck'event and ck = '0' then
          exmem.npc <= diex.npc;
          exmem.salta <= salta;
          exmem.RALU <= RALU;
          exmem.adD <= adD;
          exmem.RB <= diex.RB;
          exmem.uins <= diex.uins;
     end if;
end process;
```

```
--=============================================================================
-- MEM/ER Pipeline Register
--=============================================================================

process(ck, rst)
begin
     if rst = '1' then
          memer.npc <= (others=>'0');
          memer.RALU <= (others=>'0');
          memer.adD <= (others=>'0');
          memer.MDR <= (others=>'0');
          -- uins rst
          memer.uins.inst_branch <= '0';
          memer.uins.inst_grupol <= '0';
          memer.uins.inst_grupoI <= '0';
          memer.uins.wreg <= '0';
          memer.uins.ce <= '0';
          memer.uins.rw <= '0';
          memer.uins.bw <= '1';
          memer.uins.i <= NOP;

     elsif ck'event and ck = '0' then
          memer.npc <= exmem.npc;
          memer.RALU <= exmem.RALU;
          memer.adD <= exmem.adD;
          memer.MDR <= mdr_int;
          memer.uins <= exmem.uins;
     end if;
end process;
```

As figuras acima não exaurem o conjunto das adaptações feitas, mesmo porque uma descrição completa requereria a análise minuciosa de cada linha de código, mas servem para dar uma ideia geral das modificações realizadas.

Cumpre ressaltar, todavia, que o banco de registradores foi modificado para dar suporte ao modelo mestre-escravo, possibilitando que leitura e escrita no Banco de Registradores sejam feitas no mesmo período do ciclo de relógio – uma em nível lógico alto, outra no baixo.

Executados os programas na MIPS multiticlo e na MIPS pipeline, obtivemos os seguintes gráficos[3]:



No primeiro, estudam-se as diferentes implementações em duas curvas distintas, analisando-se sua evolução ao longo dos programas. No segundo, Os programas é que são representados em curvas, correspondendo a metade

[3] Os *prints* com os resultados dos tempos de execução podem ser encontrados no arquivo .zip em que está este relatório está inserido.

esquerda do gráfico à execução deles na implementação multiciclo e a direita à pipeline sem tratamento de conflitos.

Percebe-se que essa primeira versão da MIPS pipeline possui desempenho superior (menor tempo de execução) em todos os programas estudados e, enquanto a melhoria é discreta nos programas 1 e 3, no programa 2 (ordenação de vetores), ela é bastante significativa.