Conception avancée des systèmes informatiques
Hiver 2021

Processeur a Pipeline

Membre du groupe:

Amen Abshir - 300035421
Oumou Chérif Bah - 300036342

I- Partie theorique

1- Introduction

Dans le but d’augmenter la performance du processeur, le concepteur utilise le processus de Pipelining.

Le pipeline est un mécanisme permettant d'accroître la vitesse d'exécution des instructions dans un micro-processeur. L'idée générale est d'appliquer le principe du travail à la chaîne à l'exécution des instructions. Dans un microprocesseur sans pipeline, les instructions sont exécutées les unes après les autres. Une nouvelle instruction n'est commencée que lorsque l'instruction précédente est complètement terminée. Avec un pipeline, le micro-processeur commence une nouvelle instruction avant d'avoir fini la précédente. Plusieurs instructions se trouvent donc simultanément en cours d'exécution au cœur du micro-processeur. Le temps d'exécution d'une seule instruction n'est pas réduit. Par contre, le débit du micro-processeur, c'est-à-dire le nombre d'instructions exécutées par unité de temps, est augmenté. Il est multiplié par le nombre d'instructions qui sont exécutées simultanément

Dans ce laboratoire, nous allons implémenter un processeur pipeline en tenant compte des données disponibles dans l'énoncé du laboratoire telles que la spécification d'entrée et sorties, nombre de bit pour le chemin de donnée , la largeur des instructions et de la mémoire .

Le pipeline a implémenter comprend cinq étapes :

1) Récupérer l'instruction de la mémoire: Comme nous avons besoin que plus d'une instruction entre dans le chemin de donnée, il faut supposer que cette étape se fait en un cycle d'horloge et il faut normalement 5 cycles pour exécuter une instruction. A la fin de cette étape, toutes les instructions à exécuter sont chargées dans la pipeline chacune a une étape différente (décalage)..

2) Lire les registres tout en décodant l'instruction: Une fois les instructions récupérer, il faut les décoder et les lire dans les registres à chaque horloge de cycle
3) Exécuter l'opération ou calculer une adresse: On effectue l'opération demandée tout en ayant accès à la mémoire et aux registres
4) Selon l'opération à effectuer , on peut aller dans la mémoire de donner et prendre des opérandes u placer quelque chose dans la mémoire
5) Écrire le résultat final dans un registre, la source de résultat peut être la mémoire, l'unité de calcul ou même un autre registre.

Lors de l'exécution d'un pipeline, l'on peut rencontrer trois types de hasard causant ainsi l'échec du pipeline:

les hasards structurels lorsque la combinaison d'instruction n'est pas supporté par le chemin, les hasards de contrôle lorsqu'une décision est prise après l'exécution d'une seul instruction tandis que les autres s'exécute encore, et enfin les hasards de donnée lorsqu'une instruction dépend des résultats d'une autre instruction sur le pipeline.

Dans ce laboratoire, nous allons ignorer tout hasard structurel pouvant survenir.

2- Objectif

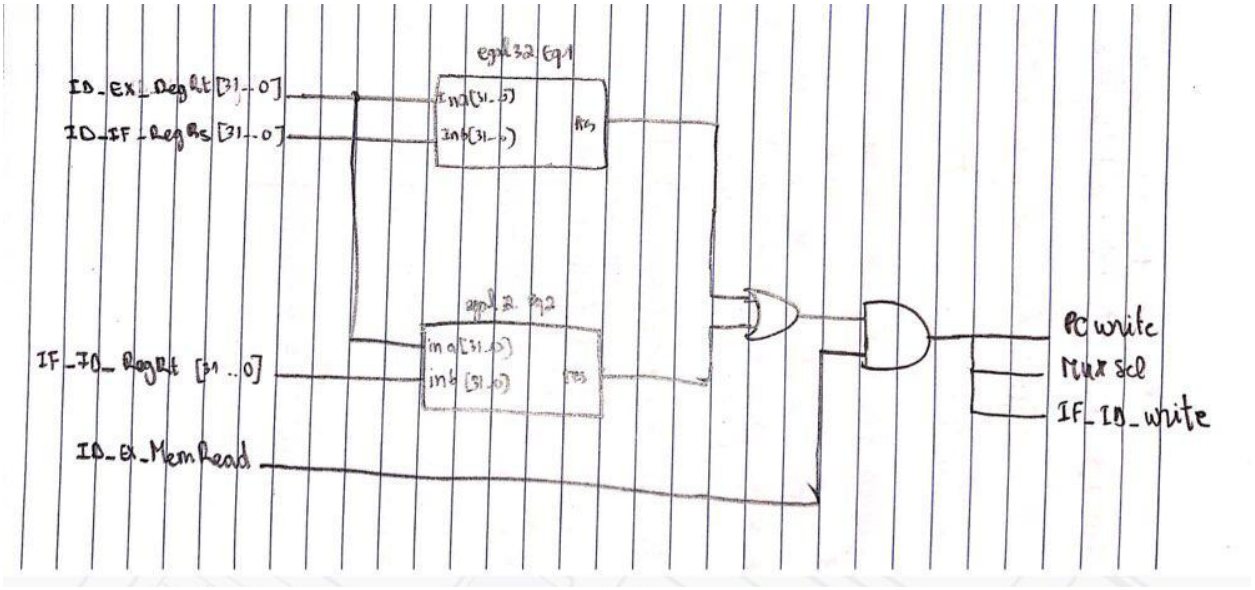
L'objectif de ce laboratoire est de concevoir et de construire un processeur pipeline RISC en VHDL. A la fin de ce laboratoire, l'étudiant(e) doit être capable de:

1. Concevoir, réaliser et de tester un processeur pipeline RISC;
2. Démontrer une compréhension complète des concepts de pipelining, y compris les hasards, dans le contexte des processeurs RISC et architectures d'ensemble d'instructions.

3- Prélab

Concevoir la détection des hasards et unités d'expédition décrits dans ce laboratoire

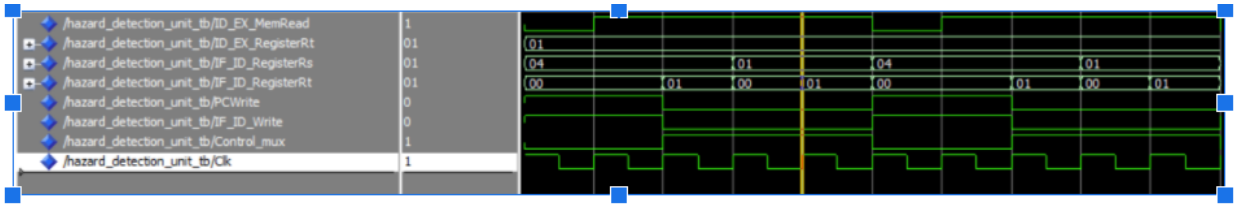
Hazard detecting
La composante de détection de hasard à 3 sorties (PCWrite, MuxSel et IF/ID.Write) la valeur de chacune de ces sorties et identique. Lorsqu'on détecte un hasard, nous devons avoir un "nop" donc les sorties sont égales à 0. Cette détection de hasard se passe lorsque ID/EX.MemRead = 1, ID/EX.RegRt est égale à IF/ID.RegRs ou IF/ID.RegRt.



```
library ieee;
use ieee.std_logic_1164.all;

entity Hazard_detection_unit is

  port (
    ID_EX_MemRead                    : in  std_logic;
    ID_EX_RegisterRt                 : in  std_logic_vector(4 downto 0);
    IF_ID_RegisterRs, IF_ID_RegisterRt : in  std_logic_vector(4 downto 0);
    PCWrite, IF_ID_Write, Control_mux  : out std_logic);

end entity Hazard_detection_unit;

architecture basic of Hazard_detection_unit is

  signal Control_mux_temp: std_logic;
  signal ID_EX_RegisterRt_eq_IF_ID_RegisterRs: std_logic;
  signal ID_EX_RegisterRt_eq_IF_ID_RegisterRt: std_logic;
begin  -- architecture basic

  ID_EX_RegisterRt_eq_IF_ID_RegisterRs<= not (ID_EX_RegisterRt(4) xor IF_ID_RegisterRs(4)) and not (ID_EX_RegisterRt(3) xor IF_ID_RegisterRs(3)) and not (
ID_EX_RegisterRt(2) xor IF_ID_RegisterRs(2)) and not (ID_EX_RegisterRt(1) xor IF_ID_RegisterRs(1)) and not (ID_EX_RegisterRt(0) xor IF_ID_RegisterRs(0));

  ID_EX_RegisterRt_eq_IF_ID_RegisterRt<= not (ID_EX_RegisterRt(4) xor IF_ID_RegisterRt(4)) and not (ID_EX_RegisterRt(3) xor IF_ID_RegisterRt(3)) and not (
ID_EX_RegisterRt(2) xor IF_ID_RegisterRt(2)) and not (ID_EX_RegisterRt(1) xor IF_ID_RegisterRt(1)) and not (ID_EX_RegisterRt(0) xor IF_ID_RegisterRt(0));



  Control_mux_temp<= ID_EX_MemRead and( ID_EX_RegisterRt_eq_IF_ID_RegisterRs or ID_EX_RegisterRt_eq_IF_ID_RegisterRt);
  Control_mux<= Control_mux_temp;
  PCWrite<=not Control_mux_temp;
  IF_ID_Write<= not Control_mux_temp;


end architecture basic;
```

Voir code a la fin

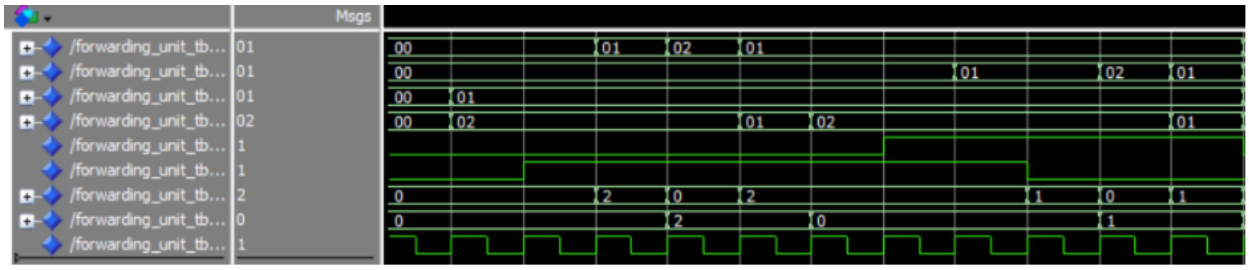
Hazard detecting simulation



Forwarding unit

Cette composante est utilisée pour détecter les hasards de données et donner la valeur de ForwardA et ForwardB pour nos deux multiplexeurs. Nous avons 6 entrées (EX_MEM_RegRd, MEM_WB_RegRd, ID_EX_RegRs, ID_EX_RegRt,EX_MEM_RegWrite, MEM_WB_RegWrite) et nos deux sorties (ForwardA et ForwardB) qui vont être égale à “10”, “01” ou “00” selon les valeurs des entrées

Forwarding unit Simulation



II-Partie conception

Alu_onebit et nbit_alu

```
-------------------------------------------------------------------------
-------------
-- Title      : One bit ALU
-- Project    :
-------------------------------------------------------------------------
-------------
-- File       : alu.vhd
-- Author     :  Amen Abshir
-- Company    :
-- Created    : 2021-03-10
-- Last update: 2021-03-17
-- Platform   :
-- Standard   : VHDL'93/02
-------------------------------------------------------------------------
-------------
-- Description: One bit ALU based on Figure B.5.10 of Computer
Organization and
-- design by David A. Patterson and John L.Hennessy
-------------------------------------------------------------------------
-------------
-- Copyright (c) 2021
-------------------------------------------------------------------------
-------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2021-03-10  1.0      DELL    Created
-------------------------------------------------------------------------
-------------


library ieee;
use ieee.std_logic_1164.all;

entity alu_onebit is

  port (
    a, b                        : in  std_logic;
    Ainvert, Binvert            : in  std_logic;  -- Control bits
    Operation                   : in  std_logic_vector(1 downto
0);
    Carryin, Less               : in  std_logic;
    CarryOut, Result,Set            : out std_logic);

end entity alu_onebit;
architecture basic  of alu_onebit is
  signal mux_1_o, mux_2_o: std_logic;
  signal not_a, not_b: std_logic;
  signal and_o, or_o, add_o: std_logic;
  component oneBitAdder is
    port (
      i_CarryIn         : IN  STD_LOGIC;
      i_Ai, i_Bi        : IN  STD_LOGIC;
      o_Sum, o_CarryOut : OUT STD_LOGIC);
  end component oneBitAdder;
  component mux4_2 is
    port (
      s1, s0         : in  std_logic;
      x0, x1, x2, x3 : in  std_logic;
      f              : out std_logic);
  end component mux4_2;
  component mux_21 is
    port (
      i_1    : in  STD_LOGIC;
      i_2    : in  STD_LOGIC;
      sel    : in  STD_LOGIC;
      output : out STD_LOGIC);
  end component mux_21;



begin  -- architecture basic
  not_a<= not a;
  not_b<= not b;
  mux_21_1: entity work.mux_21
    port map (
      i_1    => a,
      i_2    => not_a,
      sel    => Ainvert,
      output => mux_1_o);
  mux_21_2: entity work.mux_21
    port map (
      i_1    => b,
      i_2    => not_b,
      sel    => Binvert,
      output => mux_2_o);

  and_o<= mux_1_o and mux_2_o;
  or_o<= mux_1_o or mux_2_o;
  oneBitAdder_1: entity work.oneBitAdder
    port map (
      i_CarryIn  => Carryin,
      i_Ai       => mux_1_o,
      i_Bi       => mux_2_o,
      o_Sum      => add_o,
      o_CarryOut => CarryOut);
  Set<=add_o;
  mux4_2_1: entity work.mux4_2
    port map (
      s1 => Operation(1),
      s0 => Operation(0),
      x0 => and_o,
      x1 => or_o,
      x2 => add_o,
      x3 => Less,
      f  => Result);




end architecture basic ;
```

Figure2 : Screenshot du code de alu_onebit

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity nbit_ALU is

  generic (
    n : positive := 31);

  port (
    a, b             : in  std_logic_vector(n downto 0);  -- operands
    Ainvert, Binvert : in  std_logic;
    Operation        : in  std_logic_vector;
    result           : out std_logic_vector(n downto 0);
    zero             : out std_logic);

end entity nbit_ALU;

architecture basic of nbit_ALU is
  signal i_carry, i_result,temp : std_logic_vector(n downto 0);
  signal i_set, i_zero : std_logic;


  component alu_onebit is
    port (
      a, b                   : in  std_logic;
      Ainvert, Binvert       : in  std_logic;
      Operation              : in  std_logic_vector(1 downto 0);
      Carryin, Less          : in  std_logic;
      CarryOut, Result, Set : out std_logic);
  end component alu_onebit;

begin  -- architecture basic

  alu_onebit_1: entity work.alu_onebit
    port map (
      a         => a(0),
      b         => b(0),
      Ainvert   => Ainvert,
      Binvert   => Binvert,
      Operation => Operation,
      Carryin   => Binvert,
      Less      => i_set,
      CarryOut  => i_carry(0),
      Result    =>i_result(0));

  alu_gen: for i in 1 to n-1 generate
    alu_onebit_2: entity work.alu_onebit
      port map (
        a         => a(i),
        b         => b(i),
        Ainvert   => Ainvert,
        Binvert   => Binvert,
        Operation => Operation,
        Carryin   =>i_carry(i-1) ,
        Less      => '0',
        CarryOut  => i_carry(i),
        Result    => i_result(i));
  end generate alu_gen;
  alu_onebit_3: entity work.alu_onebit
    port map (
      a         => a(n),
      b         => b(n),
      Ainvert   => Ainvert,
      Binvert   => Binvert,
      Operation => Operation,
      Carryin   => i_carry(n-1),
      Less      => '0',
      CarryOut  => i_carry(n),
      Result    => i_result(n),
      Set       => i_set);

  temp(0)<=i_result(0);
```

Figure3 : Screenshot du code de nbit_alu.vhd

Le composant alu est implémenté à l'aide du composant alu onebit instancie n fois. La variable indique le nombre de bit qu'on veut utiliser dans nos opérandes. Cette entité, basée sur la figure B.10.5 dans l'annexe B du livre Computer design and organization, est capable de faire les opérations and, or, add, sub et less. Il peut aussi inverser les entrées. Il a comme sortie le résultat de l'opération et le signal de statut zero.

ALU_control

```
library ieee;
use ieee.std_logic_1164.all;

entity alu_control is

  port (
    AluOp     : in  std_logic_vector(1 downto 0);
    Funct     : in  std_logic_vector(5 downto 0);
    Operation : out std_logic_vector(2 downto 0));

end entity alu_control;

architecture basic of alu_control is
  signal op_2, op_1, op_0, i_1 : std_logic;


begin  -- architecture basic
  i_1<= AluOp(1) and Funct(1);

  Operation(2)<=i_1 or AluOp(0);
  Operation(1)<=(not (AluOp(1))) or (not Funct(2) );
  Operation(0)<=(Funct(3) or Funct(0)) and AluOp(1);




end architecture basic;
```

Code VHDL de alu_control

Cette entité nous permet de choisir l'opération de notre alu a partir du champs funct de l'instruction et le signal de control AluOp.

Clk_div

```vhdl
use ieee.std_logic_1164.all;

entity clk_div is

  port (
    GClock, GReset   : in  std_logic;
    PC_clock : out std_logic);

end entity clk_div;

architecture basic of clk_div is
  signal out_1, out_2, out_3: std_logic;
  signal i_1, i_2, i_3: std_logic;
  signal not_out_1, not_out_2, not_out_3: std_logic;
  component enARdFF_2 is
    port (
      i_resetBar  : IN  STD_LOGIC;
      i_d         : IN  STD_LOGIC;
      i_enable    : IN  STD_LOGIC;
      i_clock     : IN  STD_LOGIC;
      o_q, o_qBar : OUT STD_LOGIC);
  end component enARdFF_2;
begin  -- architecture basic
  i_1<= not_out_1 and out_2 and out_3;
  enARdFF_2_1: entity work.enARdFF_2
    port map (
      i_resetBar => GReset,
      i_d        => i_1,
      i_enable   => '1',
      i_clock    => GClock,
      o_q        => out_1,
      o_qBar     => not_out_1);
i_2<=( not_out_1 and not_out_2 and out_3)or ( not_out_1 and out_2 and  not_out_3);
  enARdFF_2_2: entity work.enARdFF_2
    port map (
      i_resetBar => GReset,
      i_d        => i_2,
      i_enable   => '1',
      i_clock    => GClock,
      o_q        => out_2,
      o_qBar     => not_out_2);
  i_3<=not_out_1 and not_out_3;
  enARdFF_2_3: entity work.enARdFF_2
    port map (
      i_resetBar => GReset,
      i_d        => i_3,
      i_enable   => '1',
      i_clock    => GClock,
      o_q        => out_3,
      o_qBar     => not_out_3);

  PC_clock <= out_1 and not(out_2) and not(out_3);

end architecture basic;
```

Figure4 : Screenshot du code de clk_div.vhd

Cette entité a comme entrée GClock et GReset et a comme sortie PC_clock. Il nous permet de diviser l'horloge globale par 5, pour que la valeur du compteur de programme (PC) s'incrémente tous les 5 impulsions de l'horloge Globale.

Controlpath

```
library ieee;
use ieee.std_logic_1164.all;

entity controlpath is

  port (
    Op                                                                  : in  std_logic_vector(5 downto 0);
    RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp1, ALUOp0, Jump : out std_logic);

end entity controlpath;

architecture basic of controlpath is
  signal r_format, lw, sw, beq, j : std_logic;

begin  -- architecture basic
  r_format<=(not Op(5)) and (not Op(4)) and (not Op(3)) and (not Op(2)) and (not Op(1)) and (not Op(0));
  lw<=(Op(5)) and (not Op(4)) and (not Op(3)) and (not Op(2)) and (Op(1)) and (Op(0));

 sw<= (Op(5)) and (not Op(4)) and (Op(3)) and (not Op(2)) and (Op(1)) and (Op(0));
  beq<= (not Op(5)) and (not Op(4)) and (not Op(3)) and ( Op(2)) and (not Op(1)) and (not Op(0));
  j<=(not Op(5)) and (not Op(4)) and (not Op(3)) and (not Op(2)) and (Op(1)) and (not Op(0));
  RegDst<=r_format;
  ALUSrc<=lw or sw;
  MemtoReg<=lw;
  RegWrite<=r_format or lw;
  MemRead<=lw;
  MemWrite<=sw;
  Branch<=beq;
  ALUOp1<=r_format;
  ALUOp0<=beq;
  Jump<=j;

end architecture basic;
```

Figure5 : Screenshot de l'entite controlpath

Controlpath prend comme entrée Op, qui correspond au 6 premier bit de l'instruction et a comme sortie les signaux de contrôles nécessaires pour le fonctionnement de notre processeur. Il est implémenté comme un décodeur simple avec des portes ***et*** ayant un fan-in de 6.

Data_memory

```
-- megafunction wizard: %RAM: 2-PORT%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altsyncram

-- ============================================================
-- File Name: data_memory.vhd
-- Megafunction Name(s):
--                      altsyncram
--
-- Simulation Library Files(s):
--                      altera_mf
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!

-- 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
-- ************************************************************


--Copyright (C) 1991-2013 Altera Corporation
--Your use of Altera Corporation's design tools, logic functions
--and other software and tools, and its AMPP partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Altera Program License
--Subscription Agreement, Altera MegaCore Function License
--Agreement, or other applicable license agreement, including,
--without limitation, that your use is for the sole purpose of
--programming logic devices manufactured by Altera and sold by
--Altera or its authorised distributors.  Please refer to the
--applicable agreement for further details.


LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY data_memory IS
        PORT
        (
                clock           : IN STD_LOGIC  := '1';
                data            : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                rdaddress               : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                rden            : IN STD_LOGIC  := '1';
                wraddress               : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                wren            : IN STD_LOGIC  := '0';
                q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
        );
END data_memory;
```

Figure6 : Screenshot partiel de l'entite data_memory.vhd

Cette entité correspond à notre data memory. Il a été implémenté à l'aide la fonction Megafunction wizard de Quartus. C'est un ram ainsi on est capable de lire son contenu q en provenant l'address de lecture rdaddress et le signal de control rden ou écrire dans

les cellules de mémoire en provenant l’address d'écriture wraddress et le signal de control wren.

Instruction Memory

```
-- megafunction wizard: %ROM: 1-PORT%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altsyncram

-- ============================================================
-- File Name: instruction_memory.vhd
-- Megafunction Name(s):
--                      altsyncram
--
-- Simulation Library Files(s):
--                      altera_mf
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
-- ************************************************************


--Copyright (C) 1991-2013 Altera Corporation
--Your use of Altera Corporation's design tools, logic functions
--and other software and tools, and its AMPP partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Altera Program License
--Subscription Agreement, Altera MegaCore Function License
--Agreement, or other applicable license agreement, including,
--without limitation, that your use is for the sole purpose of
--programming logic devices manufactured by Altera and sold by
--Altera or its authorized distributors.  Please refer to the
--applicable agreement for further details.


LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY instruction_memory IS
        PORT
        (
                address         : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                clock           : IN STD_LOGIC  := '1';
                q               : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
        );
END instruction_memory;
```

Figure7 : Screenshot partiel de l'entité instruction_memory.vhd

Cette entité correspond à notre data memory. Il a été implémenté à l'aide la fonction Megafunction wizard de Quartus. C'est un ROM ainsi on est capable de lire son contenu q en provenant l'adresse de lecture. Cette adresse de lecture est PC.

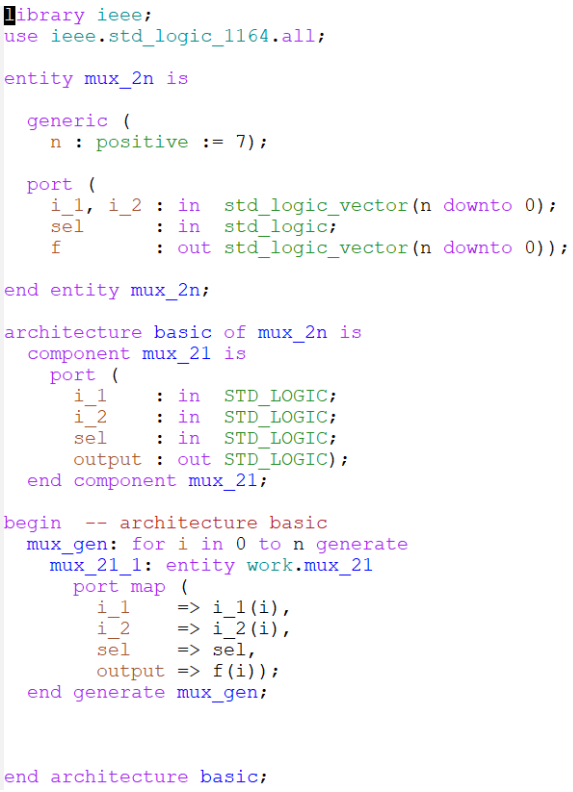
Mux_21

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity mux_21 is
  port(i_1 : in STD_LOGIC;
       i_2 : in STD_LOGIC;
       sel:  in STD_LOGIC;
       output: out STD_LOGIC);
end mux_21;

architecture basic of mux_21 is

begin  -- architecture basic

  output <=( i_1 and not (sel)) or (i_2 and sel);

end architecture basic;
```

Figure8

Cette entité prend est un multiplexeur qui prend comme entrée 2 bits et un sélecteur de 1 bit.

Mux_2n



```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mux_2n is

  generic (
    n : positive := 7);

  port (
    i_1, i_2 : in  std_logic_vector(n downto 0);
    sel      : in  std_logic;
    f        : out std_logic_vector(n downto 0));

end entity mux_2n;

architecture basic of mux_2n is
  component mux_21 is
    port (
      i_1    : in  STD_LOGIC;
      i_2    : in  STD_LOGIC;
      sel    : in  STD_LOGIC;
      output : out STD_LOGIC);
  end component mux_21;

begin  -- architecture basic
  mux_gen: for i in 0 to n generate
    mux_21_1: entity work.mux_21
      port map (
        i_1    => i_1(i),
        i_2    => i_2(i),
        sel    => sel,
        output => f(i));
  end generate mux_gen;


end architecture basic;
```

Figure9: Screenshot de l'entite mux2n

Cette entité prend est un multiplexeur qui prend comme entrée 2 vecteurs de n bits et un sélecteur de 1 bit. On utilise le composant mux_21 pour l'implémentation.

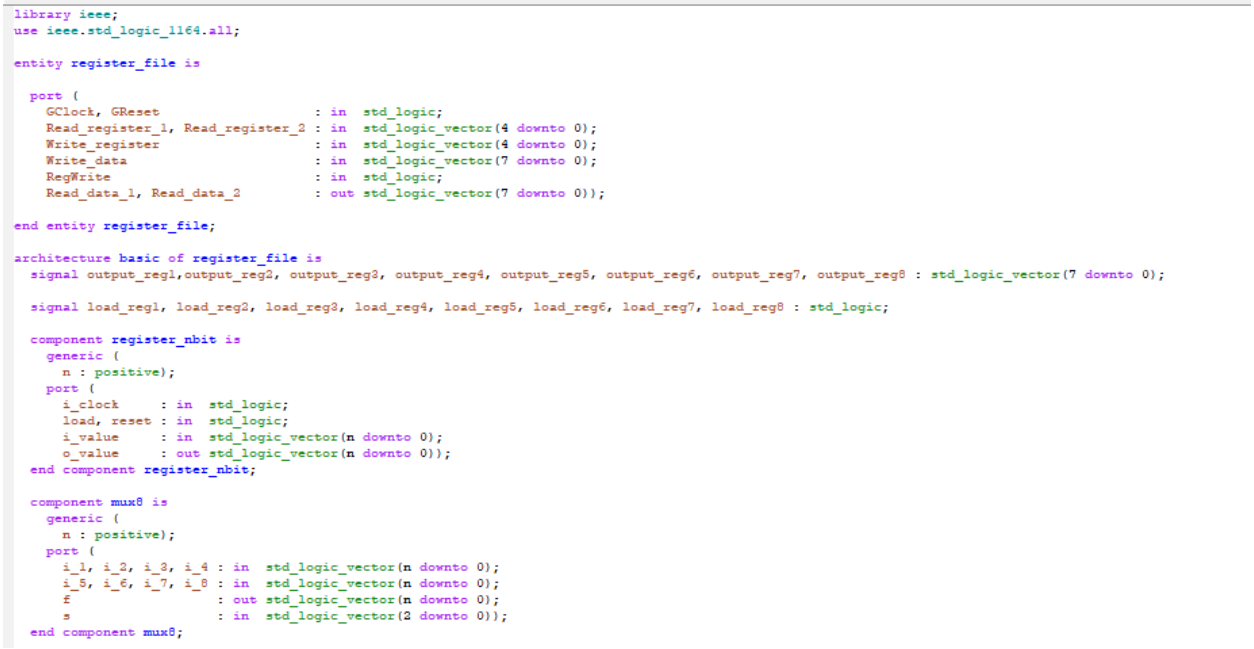
Mux8

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity mux8 is

  generic (
    n : positive := 7);

  port (
    i_1, i_2, i_3,i_4  : in  std_logic_vector(n downto 0);
    i_5, i_6, i_7,i_8 : in  std_logic_vector(n downto 0);
    f                 : out std_logic_vector(n downto 0);
    s                 : in  std_logic_vector(2 downto 0));

end entity mux8;

architecture basic of mux8 is
  signal to_mux1,to_mux2, to_mux3, to_mux4, to_mux5, to_mux6: std_logic_vector(n downto 0);

begin
  mux_gen: for i in 0 to n generate

    mux_21_1: entity work.mux_21
      port map (
        i_1    => i_1(i),
        i_2    => i_2(i),
        sel    => s(0),
        output => to_mux1(i));
     mux_21_2: entity work.mux_21
      port map (
        i_1    => i_3(i),
        i_2    => i_4(i),
        sel    => s(0),
        output => to_mux2(i));
     mux_21_3: entity work.mux_21
      port map (
        i_1    => i_5(i),
        i_2    => i_6(i),
        sel    => s(0),
        output => to_mux3(i));
     mux_21_4: entity work.mux_21
      port map (
        i_1    => i_7(i),
        i_2    => i_8(i),
        sel    => s(0),
        output => to_mux4(i));

    mux_21_5: entity work.mux_21
      port map (
        i_1    => to_mux1(i),
        i_2    => to_mux2(i),
        sel    => s(1),
        output => to_mux5(i));

     mux_21_6: entity work.mux_21
      port map (
        i_1    => to_mux3(i),
        i_2    => to_mux4(i),
        sel    => s(1),
        output => to_mux6(i));
    mux_21_7: entity work.mux_21
      port map (
        i_1    => to_mux5(i),
        i_2    => to_mux6(i),
        sel    => s(2),
        output => f(i));


  end generate mux_gen;
end architecture basic;
```

Figure10 : Screenshot du code de mux8.vhd

Cette entité prend est un multiplexeur qui prend comme entrée 8 vecteurs de n bits et un sélecteur de 3 bit. On utilise le composant mux_2n pour l'implémentation.

Register_file



```
library ieee;
use ieee.std_logic_1164.all;

entity register_file is

  port (
    GClock, GReset                    : in  std_logic;
    Read_register_1, Read_register_2 : in  std_logic_vector(4 downto 0);
    Write_register                    : in  std_logic_vector(4 downto 0);
    Write_data                        : in  std_logic_vector(7 downto 0);
    RegWrite                          : in  std_logic;
    Read_data_1, Read_data_2          : out std_logic_vector(7 downto 0));

end entity register_file;

architecture basic of register_file is
  signal output_reg1,output_reg2, output_reg3, output_reg4, output_reg5, output_reg6, output_reg7, output_reg8 : std_logic_vector(7 downto 0);

  signal load_reg1, load_reg2, load_reg3, load_reg4, load_reg5, load_reg6, load_reg7, load_reg8 : std_logic;

  component register_nbit is
    generic (
      n : positive);
    port (
      i_clock     : in  std_logic;
      load, reset : in  std_logic;
      i_value     : in  std_logic_vector(n downto 0);
      o_value     : out std_logic_vector(n downto 0));
  end component register_nbit;

  component mux8 is
    generic (
      n : positive);
    port (
      i_1, i_2, i_3, i_4 : in  std_logic_vector(n downto 0);
      i_5, i_6, i_7, i_8 : in  std_logic_vector(n downto 0);
      f                  : out std_logic_vector(n downto 0);
      s                  : in  std_logic_vector(2 downto 0));
  end component mux8;
```

Figure 11 : Screenshot partiel de l'entité register file.vhd

Cette entité correspond au fichier de registre de notre processeur. Il contient 8 registres de 8 bit. La décision de lire quel registre et mettre sa valeur dans les sorties Read_data_1 ou Read_data_2 dépend de la valeur des champs rs et rd de l'instruction. Ces champs correspondent aux entrée Read_register_1 et Read_regiser_2. La décision d'écrire dans quel registre dépend de la valeur du champ rt de l'instruction et le signal de contrôle RegWrite. Ce champ correspond à l'entrée Write_register. Write data est écrit dans Write_register.

Register_nbit

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity register_nbit  is

  generic (
    n : positive := 7);

  port (
    i_clock     : in  std_logic;
    load, reset : in  std_logic;
    i_value     : in  std_logic_vector(n downto 0);
    o_value     : out std_logic_vector(n downto 0));

end entity register_nbit ;

architecture basic of register_nbit is
  component enARdFF_2 is
    port (
      i_resetBar  : IN  STD_LOGIC;
      i_d         : IN  STD_LOGIC;
      i_enable    : IN  STD_LOGIC;
      i_clock     : IN  STD_LOGIC;
      o_q, o_qBar : OUT STD_LOGIC);
  end component enARdFF_2;


begin  -- architecture basic
  loopn: for i in 0 to n generate
    enARdFF_2_1: entity work.enARdFF_2
      port map (
        i_resetBar => reset,
        i_d        => i_value(i),
        i_enable   => load,
        i_clock    => i_clock,
        o_q        => o_value(i));
  end generate loopn;



end architecture basic;
```

Figure11: Code VHDL de registrer_nbit .

Cette entite est un registre a n bit.

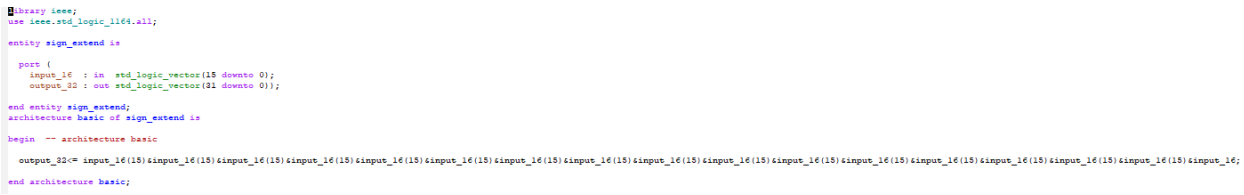
Shift_left_nbit.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity shift_left_2 is

  generic (
    n : positive := 7);                 -- entrance bit count

  port (
    input_n       : in  std_logic_vector(n downto 0);
    output_shift2 : out std_logic_vector(n+2 downto 0));

end entity shift_left_2;

architecture basic of shift_left_2 is
  begin

  output_shift2<= input_n & "00";

end architecture basic;
```

Figure12 :Code vhdl de shift_left_2

Cette entité prend comme entrée un vecteur de n bit et ajoute “00” comme lsb.

Sign_extend



```
library ieee;
use ieee.std_logic_1164.all;

entity sign_extend is

  port (
    input_16  : in  std_logic_vector(15 downto 0);
    output_32 : out std_logic_vector(31 downto 0));

end entity sign_extend;
architecture basic of sign_extend is

begin  -- architecture basic

  output_32<= input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16(15)&input_16;

end architecture basic;
```

Figure13 Code Vhdl de sign_extend.

Cette entité prend comme entrée un vecteur de 16 bit et l'étend a 32 bit sans changer le signe original.

Equal test

```
library ieee;
use ieee.std_logic_1164.all;

entity Equal_test is

  port (
    Read_data_1, Read_data_2 : in  std_logic_vector(7 downto 0);
    status_equal             : out std_logic);

end entity Equal_test;

architecture basic of Equal_test is

begin  -- architecture basic

  status_equal<= not( Read_data_1(7) xor Read_data_2(7)) and not( Read_data_1(6) xor Read_data_2(6)) and not( Read_data_1(5) xor Read_data_2(5)) and not( Read_data_1(4) xor Read_data_2(4)) and not( Read_data_1(3) xor Read_data_2(3)) and not( Read_data_1(2) xor Read_data_2(2)) and not( Read_data_1(1) xor Read_data_2(1)) and not( Read_data_1(0) xor Read_data_2(0));

end architecture basic;
```

Figure Code Vhdl de Equal_test.

Cette entité prend comme entrée deux vecteurs de 8 bit et test si ils sont égaux. On fait cela en opérant un not-xor bit par bit. La sortie est 1 s’ils sont égaux, sinon c’est 0.

IF/ID

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity IF_ID is

  port (
    instruction     : in  std_logic_vector(31 downto 0);
    pc_inc          : in  std_logic_vector(7 downto 0);
    GClock, GReset,IF_ID_Write  : in  std_logic;
    instruction_out : out std_logic_vector(31 downto 0);
    pc_inc_out      : out std_logic_vector(7 downto 0));
end entity IF_ID;

architecture basic of IF_ID is
  component register_nbit is
    generic (
      n : positive);
    port (
      i_clock     : in  std_logic;
      load, reset : in  std_logic;
      i_value     : in  std_logic_vector(n downto 0);
      o_value     : out std_logic_vector(n downto 0));
  end component register_nbit;

begin  -- architecture basic

  instruction_reg: entity work.register_nbit
    generic map (
      n => 31)
    port map (
      i_clock => GClock,
      load    => IF_ID_Write,
      reset   => GReset,
      i_value => instruction,
      o_value => instruction_out);

  register_nbit_1: entity work.register_nbit
    generic map (
      n => 7)
    port map (
      i_clock => GClock,
      load    => IF_ID_Write,
      reset   => GReset,
      i_value => pc_inc,
      o_value => pc_inc_out);



end architecture basic;
```

Figure Code Vhdl de IF_ID.

Cette entité correspond prend comme entrée le chemin de donne obtenue à l'étape extraction d'instructions (IF) du processeur et le transfert à l'étape de décodage d'instructions.

ID/EX

```
library ieee;
use ieee.std_logic_1164.all;

entity ID_EX is

  port (

    ALUSrc, RegDst                          : in std_logic;
    ALUOp    : in std_logic_vector(1 downto 0);
    GClock, GReset                          : in std_logic;
    Branch, MemRead, MemWrite               : in std_logic;
    RegWrite, MemToReg                      : in std_logic;
    PC                                      : in std_logic_vector(7 downto 0);
    Read_data_1, Read_data_2                : in std_logic_vector(7 downto 0);
    sign_extend                             : in std_logic_vector(31 downto 0);
    instruction_20_16, instruction_15_11    : in std_logic_vector(4 downto 0);
    ALUSrc_out, RegDst_out            : out std_logic;
    ALUOp_out    : out std_logic_vector(1 downto 0);
    Branch_out, MemRead_out, MemWrite_out   : out std_logic;
    RegWrite_out, MemToReg_out              : out  std_logic;
    PC_out                                  : out  std_logic_vector(7 downto 0);
    Read_data_1_out, Read_data_2_out        : out  std_logic_vector(7 downto 0);
    sign_extend_out                         : out  std_logic_vector(31 downto 0);
    instruction_20_16_out, instruction_15_11_out : out std_logic_vector(4 downto 0));

end entity ID_EX;

architecture basic of ID_EX is
  signal EX: std_logic_vector(3 downto 0);
  signal M: std_logic_vector(2 downto 0);
  signal WB: std_logic_vector(1 downto 0);
  signal EX_out: std_logic_vector(3 downto 0);
  signal M_out: std_logic_vector(2 downto 0);
  signal WB_out: std_logic_vector(1 downto 0);

  component register_nbit is
    generic (
      n : positive);
    port (
      i_clock     : in  std_logic;
      load, reset : in  std_logic;
      i_value     : in  std_logic_vector(n downto 0);
      o_value     : out std_logic_vector(n downto 0));
  end component register_nbit;
begin  -- architecture basic

  EX<= ALUSrc&ALUOp&RegDst;
  EX_reg: entity work.register_nbit
    generic map (
      n => 3)
    port map (
      i_clock => GClock,
      load    => '1',
      reset   => GReset,
      i_value => EX,
      o_value => EX_out);
```

Figure Code Vhdl de ID_EX.

Cette entité correspond prend comme entrée le chemin de donne obtenue à l'étape décodage d'instructions (ID) du processeur et le transfert à l'étape d'exécution.

EX/MEM

```
library ieee;
use ieee.std_logic_1164.all;

entity EX_MEM is

  port (


    GClock, GReset                                  : in std_logic;
    Branch, MemRead, MemWrite,Zero                      : in std_logic;
    RegWrite, MemToReg                              : in std_logic;
    PC                                              : in std_logic_vector(7 downto 0);
    ALU, Read_d2                    : in std_logic_vector(7 downto 0);
    wrb         : in std_logic_vector(4 downto 0);
    Branch_out, MemRead_out, MemWrite_out,Zero_out       : out std_logic;
    RegWrite_out, MemToReg_out                      : out  std_logic;
    PC_out                                          : out  std_logic_vector(7 downto 0);
    ALU_out, Read_d2_out            : out  std_logic_vector(7 downto 0);
    wrb_out: out std_logic_vector(4 downto 0));

end entity EX_MEM;

architecture basic of EX_MEM is
  signal M: std_logic_vector(2 downto 0);
  signal WB: std_logic_vector(1 downto 0);
  signal M_out: std_logic_vector(2 downto 0);
  signal WB_out: std_logic_vector(1 downto 0);
  component enARdFF_2 is
    port (
      i_resetBar  : IN  STD_LOGIC;
      i_d         : IN  STD_LOGIC;
      i_enable    : IN  STD_LOGIC;
      i_clock     : IN  STD_LOGIC;
      o_q, o_qBar : OUT STD_LOGIC);
  end component enARdFF_2;
  component register_nbit is
    generic (
      n : positive);
    port (
      i_clock     : in  std_logic;
      load, reset : in  std_logic;
      i_value     : in  std_logic_vector(n downto 0);
      o_value     : out std_logic_vector(n downto 0));
  end component register_nbit;
begin  -- architecture basic


  M<=Branch& MemRead&MemWrite;
  M_reg: entity work.register_nbit
    generic map (
      n => 2)
    port map (
      i_clock => GClock,
      load    => '1',
      reset   => GReset,
      i_value => M,
      o_value => M_out);
```

Figure: Code Vhdl de EX_MEM.

Cette entité correspond prend comme entrée le chemin de donne obtenue à l'étape d'exécution d'instructions (EX) du processeur et le transfert à l'étape mémoire (MEM).

MEM/WEB

```
library ieee;
use ieee.std_logic_1164.all;

entity MEM_WB is

  port (


    GClock, GReset : in std_logic;

    RegWrite, MemToReg : in std_logic;
    ReadData           : in std_logic_vector(7 downto 0);
    ALU                : in std_logic_vector(7 downto 0);
    wrb                : in std_logic_vector(4 downto 0);

    RegWrite_out, MemToReg_out : out std_logic;
    ReadData_out               : out std_logic_vector(7 downto 0);
    ALU_out                    : out std_logic_vector(7 downto 0);
    wrb_out                    : out std_logic_vector(4 downto 0));

end entity MEM_WB;

architecture basic of MEM_WB is

  signal WB     : std_logic_vector(1 downto 0);
  signal WB_out : std_logic_vector(1 downto 0);

  component register_nbit is
    generic (
      n : positive);
    port (
      i_clock     : in  std_logic;
      load, reset : in  std_logic;
      i_value     : in  std_logic_vector(n downto 0);
      o_value     : out std_logic_vector(n downto 0));
  end component register_nbit;
begin  -- architecture basic
```

Figure Code Vhdl de MEM_WB.

Cette entité correspond prend comme entrée le chemin de donne obtenue à l'étape mémoire (MEM) du processeur et le transfert à l'étape mémoire (WB).

Hazard detection unit

```
library ieee;
use ieee.std_logic_1164.all;

entity Hazard_detection_unit is

  port (
    ID_EX_MemRead                    : in  std_logic;
    ID_EX_RegisterRt                 : in  std_logic_vector(4 downto 0);
    IF_ID_RegisterRs, IF_ID_RegisterRt : in  std_logic_vector(4 downto 0);
    PCWrite, IF_ID_Write, Control_mux  : out std_logic);

end entity Hazard_detection_unit;

architecture basic of Hazard_detection_unit is

  signal Control_mux_temp: std_logic;
  signal ID_EX_RegisterRt_eq_IF_ID_RegisterRs: std_logic;
  signal ID_EX_RegisterRt_eq_IF_ID_RegisterRt: std_logic;
begin  -- architecture basic

  ID_EX_RegisterRt_eq_IF_ID_RegisterRs<= not (ID_EX_RegisterRt(4) xor IF_ID_RegisterRs(4)) and not (ID_EX_RegisterRt(3) xor IF_ID_RegisterRs(3)) and not ( 
ID_EX_RegisterRt(2) xor IF_ID_RegisterRs(2)) and not (ID_EX_RegisterRt(1) xor IF_ID_RegisterRs(1)) and not (ID_EX_RegisterRt(0) xor IF_ID_RegisterRs(0));

  ID_EX_RegisterRt_eq_IF_ID_RegisterRt<= not (ID_EX_RegisterRt(4) xor IF_ID_RegisterRt(4)) and not (ID_EX_RegisterRt(3) xor IF_ID_RegisterRt(3)) and not ( 
ID_EX_RegisterRt(2) xor IF_ID_RegisterRt(2)) and not (ID_EX_RegisterRt(1) xor IF_ID_RegisterRt(1)) and not (ID_EX_RegisterRt(0) xor IF_ID_RegisterRt(0));



  Control_mux_temp<= ID_EX_MemRead and( ID_EX_RegisterRt_eq_IF_ID_RegisterRs or ID_EX_RegisterRt_eq_IF_ID_RegisterRt);
  Control_mux<= Control_mux_temp;
  PCWrite<=not Control_mux_temp;
  IF_ID_Write<= not Control_mux_temp;


end architecture basic;
```

Figure Code Vhdl de Hazard_detection_unit.

Cette entité nous permet de détecter les hasards de contrôle et ensuite d’effectuer un décrochage ou un rinçage du tampon IF/ID.

Forwarding unit

```
library ieee;
use ieee.std_logic_1164.all;

entity Forwarding_Unit is

  port (
    EX_MEM_RegisterRd, MEM_WB_RegisterRd : in  std_logic_vector(4 downto 0);
    ID_EX_RegisterRs, ID_EX_RegisterRt   : in  std_logic_vector(4 downto 0);
    MEM_WB_RegWrite, EX_MEM_RegWrite     : in  std_logic;
    ForwardA, ForwardB                   : out std_logic_vector(1 downto 0));

end entity Forwarding_Unit;

architecture basic of Forwarding_Unit is
  signal not_zero_EX_MEM_RegisterRd: std_logic;
  signal not_zero_MEM_WB_RegisterRd: std_logic;

  signal EX_MEM_RegisterRd_eq_ID_EX_RegisterRs: std_logic;
  signal EX_MEM_RegisterRd_eq_ID_EX_RegisterRt: std_logic;

  signal MEM_WB_RegisterRd_eq_ID_EX_RegisterRs: std_logic;
  signal MEM_WB_RegisterRd_eq_ID_EX_RegisterRt: std_logic;

  signal ForwardA_temp: std_logic_vector(1 downto 0);
  signal ForwardB_temp: std_logic_vector(1 downto 0);



begin  -- architecture basic
  not_zero_EX_MEM_RegisterRd<=EX_MEM_RegisterRd(4) or EX_MEM_RegisterRd(3) or EX_MEM_RegisterRd(2) or EX_MEM_RegisterRd(1) or EX_MEM_RegisterRd(0);
  not_zero_MEM_WB_RegisterRd<=MEM_WB_RegisterRd(4) or MEM_WB_RegisterRd(3) or MEM_WB_RegisterRd(2) or MEM_WB_RegisterRd(1) or MEM_WB_RegisterRd(0);

  EX_MEM_RegisterRd_eq_ID_EX_RegisterRs<= not ( EX_MEM_RegisterRd(4) xor ID_EX_RegisterRs(4)) and  not ( EX_MEM_RegisterRd(3) xor ID_EX_RegisterRs(3)) an
d  not ( EX_MEM_RegisterRd(2) xor ID_EX_RegisterRs(2)) and  not ( EX_MEM_RegisterRd(1) xor ID_EX_RegisterRs(1)) and  not ( EX_MEM_RegisterRd(0) xor ID_EX
_RegisterRs(0));

  EX_MEM_RegisterRd_eq_ID_EX_RegisterRt<= not ( EX_MEM_RegisterRd(4) xor ID_EX_RegisterRt(4)) and  not ( EX_MEM_RegisterRd(3) xor ID_EX_RegisterRt(3)) an
d  not ( EX_MEM_RegisterRd(2) xor ID_EX_RegisterRt(2)) and  not ( EX_MEM_RegisterRd(1) xor ID_EX_RegisterRt(1)) and  not ( EX_MEM_RegisterRd(0) xor ID_EX
_RegisterRt(0));

  MEM_WB_RegisterRd_eq_ID_EX_RegisterRs<= not ( MEM_WB_RegisterRd(4) xor ID_EX_RegisterRs(4)) and  not ( MEM_WB_RegisterRd(3) xor ID_EX_RegisterRs(3)) an
d  not ( MEM_WB_RegisterRd(2) xor ID_EX_RegisterRs(2)) and  not ( MEM_WB_RegisterRd(1) xor ID_EX_RegisterRs(1)) and  not ( MEM_WB_RegisterRd(0) xor ID_EX
_RegisterRs(0));

  MEM_WB_RegisterRd_eq_ID_EX_RegisterRt<= not ( MEM_WB_RegisterRd(4) xor ID_EX_RegisterRt(4)) and  not ( MEM_WB_RegisterRd(3) xor ID_EX_RegisterRt(3)) an
d  not ( MEM_WB_RegisterRd(2) xor ID_EX_RegisterRt(2)) and  not ( MEM_WB_RegisterRd(1) xor ID_EX_RegisterRt(1)) and  not ( MEM_WB_RegisterRd(0) xor ID_EX
_RegisterRt(0));



  ForwardA_temp(1)<= EX_MEM_RegWrite and not_zero_EX_MEM_RegisterRd and EX_MEM_RegisterRd_eq_ID_EX_RegisterRs;
  ForwardB_temp(1)<= EX_MEM_RegWrite and not_zero_EX_MEM_RegisterRd and EX_MEM_RegisterRd_eq_ID_EX_RegisterRt;

  ForwardA_temp(0)<= MEM_WB_RegWrite and not_zero_MEM_WB_RegisterRd and MEM_WB_RegisterRd_eq_ID_EX_RegisterRs and not ForwardA_temp(1);
  ForwardB_temp(0)<= MEM_WB_RegWrite and not_zero_MEM_WB_RegisterRd and MEM_WB_RegisterRd_eq_ID_EX_RegisterRt and not ForwardB_temp(1);


  ForwardA<= ForwardA_temp;
  ForwardB<= ForwardB_temp;




end architecture basic;
```

Figure Code Vhdl de Forwarding_unit

Cette entité nous permet de détecter les hasards de donnée et ensuite d'expédier les bonnes valeurs à l'aide de ForwardA et ForwardB.

Pipelinedproc

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity pipelinedProc is

  port (
    GClock, GReset                                  : in  std_logic;
    ValueSelect,InstrSelect,DebugSelect             : in  std_logic_vector(2 downto 0);
    MuxOut                                          : out std_logic_vector(7 downto 0);
    InstructionOut                                  : out std_logic_vector(31 downto 0);
    DebugOut                                        : out std_logic_vector(3 downto 0);
    I_26_21, I_20_16, I_15_11                       : out std_logic_vector(4 downto 0);
    Wr, Rd1, Rd2                                      : out std_logic_vector(7 downto 0);
    BranchOut, ZeroOut, MemWriteOut, RegWriteOut : out std_logic);

end entity pipelinedProc;


architecture basic of pipelinedProc is
  ------------------------BLOCK 1----------------------------
  signal pc_input, pc_output : STD_LOGIC_VECTOR (7 DOWNTO 0);
  signal instruction         : STD_LOGIC_VECTOR (31 DOWNTO 0);
  signal Alu_out1            : STD_LOGIC_VECTOR (7 DOWNTO 0);
  signal instruction_out     : std_logic_vector(31 downto 0);
  signal pc_inc_out          : std_logic_vector(7 downto 0);
  signal GClock_2,s            : std_logic;

  component enARdFF_2 is
    port (
      i_resetBar  : IN  STD_LOGIC;
      i_d         : IN  STD_LOGIC;
      i_enable    : IN  STD_LOGIC;
      i_clock     : IN  STD_LOGIC;
      o_q, o_qBar : OUT STD_LOGIC);
  end component enARdFF_2;

  component register_nbit is
    generic (
      n : positive);
    port (
      i_clock     : in  std_logic;
      load, reset : in  std_logic;
      i_value     : in  std_logic_vector(n downto 0);
      o_value     : out std_logic_vector(n downto 0));
  end component register_nbit;
```

```
component nbit_ALU is
  generic (
    n : positive);
  port (
    a, b             : in  std_logic_vector(n downto 0);
    Ainvert, Binvert : in  std_logic;
    Operation        : in  std_logic_vector(1 downto 0);
    result           : out std_logic_vector(n downto 0);
    zero             : out std_logic);
end component nbit_ALU;


component instruction_memory is
  port (
    address : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
    clock   : IN  STD_LOGIC := '1';
    q       : OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
end component instruction_memory;



component mux_2n is
  generic (
    n : positive);
  port (
    i_1, i_2 : in  std_logic_vector(n downto 0);
    sel      : in  std_logic;
    f        : out std_logic_vector(n downto 0));
end component mux_2n;

component IF_ID is
  port (
    instruction     : in  std_logic_vector(31 downto 0);
    pc_inc          : in  std_logic_vector(7 downto 0);
    GClock, GReset,IF_ID_Write  : in  std_logic;
    instruction_out : out std_logic_vector(31 downto 0);
    pc_inc_out      : out std_logic_vector(7 downto 0));
end component IF_ID;
```

```
  ------------------------BLOCK 1----------------------------
  signal pc_input, pc_output : STD_LOGIC_VECTOR (7 DOWNTO 0);
  signal instruction         : STD_LOGIC_VECTOR (31 DOWNTO 0);
  signal Alu_out1            : STD_LOGIC_VECTOR (7 DOWNTO 0);
  signal instruction_out     : std_logic_vector(31 downto 0);
  signal pc_inc_out          : std_logic_vector(7 downto 0);
  signal GClock_2,s            : std_logic;

  component enARdFF_2 is
    port (
      i_resetBar  : IN  STD_LOGIC;
      i_d         : IN  STD_LOGIC;
      i_enable    : IN  STD_LOGIC;
      i_clock     : IN  STD_LOGIC;
      o_q, o_qBar : OUT STD_LOGIC);
  end component enARdFF_2;

  component register_nbit is
    generic (
      n : positive);
    port (
      i_clock     : in  std_logic;
      load, reset : in  std_logic;
      i_value     : in  std_logic_vector(n downto 0);
      o_value     : out std_logic_vector(n downto 0));
  end component register_nbit;

  component nbit_ALU is
    generic (
      n : positive);
    port (
      a, b             : in  std_logic_vector(n downto 0);
      Ainvert, Binvert : in  std_logic;
      Operation        : in  std_logic_vector(1 downto 0);
      result           : out std_logic_vector(n downto 0);
      zero             : out std_logic);
  end component nbit_ALU;


  component instruction_memory is
    port (
      address : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
      clock   : IN  STD_LOGIC := '1';
      q       : OUT STD_LOGIC_VECTOR (31 DOWNTO 0));
  end component instruction_memory;



  component mux_2n is
    generic (
      n : positive);
    port (
      i_1, i_2 : in  std_logic_vector(n downto 0);
      sel      : in  std_logic;
      f        : out std_logic_vector(n downto 0));
  end component mux_2n;

  component IF_ID is
    port (
      instruction     : in  std_logic_vector(31 downto 0);
      pc_inc          : in  std_logic_vector(7 downto 0);
      GClock, GReset,IF_ID_Write  : in  std_logic;
      instruction_out : out std_logic_vector(31 downto 0);
      pc_inc_out      : out std_logic_vector(7 downto 0));
  end component IF_ID;
```

```
----------------------BLOCK 2-----------------------------------

signal Read_data_1_in, Read_data_2_in : std_logic_vector(7 downto 0);
signal sign_extend_32_in : std_logic_vector(31 downto 0);

signal ALUSrc_out, RegDst_out                        : std_logic;
signal ALUOp, ALUOp_out                                    : std_logic_vector(1 downto 0);
signal Branch_out, MemRead_out, MemWrite_out         : std_logic;
signal RegWrite_out, MemToReg_out                    : std_logic;
signal PC_out                                        : std_logic_vector(7 downto 0);
signal Read_data_1_out, Read_data_2_out              : std_logic_vector(7 downto 0);
signal sign_extend_out                               : std_logic_vector(31 downto 0);
signal instruction_25_21_out, instruction_20_16_out, instruction_15_11_out : std_logic_vector(4 downto 0);

signal RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp1, ALUOp0, Jump : std_logic;

signal control_mux_in, control_mux_out : std_logic_vector(9 downto 0);
signal status_equal             : std_logic;
signal IF_Flush                 : std_logic;

component register_file is
  port (
    GClock, GReset                  : in  std_logic;
    Read_register_1, Read_register_2 : in  std_logic_vector(4 downto 0);
    Write_register                  : in  std_logic_vector(4 downto 0);
    Write_data                      : in  std_logic_vector(7 downto 0);
    RegWrite                        : in  std_logic;
    Read_data_1, Read_data_2        : out std_logic_vector(7 downto 0));
end component register_file;

component controlpath is
  port (
    Op                                                                                  : in  std_logic_vector(5 downto 0);
    RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp1, ALUOp0, Jump : out std_logic);
end component controlpath;


component sign_extend is
  port (
    input_16  : in  std_logic_vector(15 downto 0);
    output_32 : out std_logic_vector(31 downto 0));
end component sign_extend;


component Equal_test is
  port (
    Read_data_1, Read_data_2 : in  std_logic_vector(7 downto 0);
    status_equal             : out std_logic);
end component Equal_test;


component ID_EX is
  port (
    ALUSrc, RegDst                              : in  std_logic;
    ALUOp                                       : in  std_logic_vector(1 downto 0);
    GClock, GReset                              : in  std_logic;
    Branch, MemRead, MemWrite                   : in  std_logic;
    RegWrite, MemToReg                          : in  std_logic;
    PC                                          : in  std_logic_vector(7 downto 0);
    Read_data_1, Read_data_2                    : in  std_logic_vector(7 downto 0);
    sign_extend                                 : in  std_logic_vector(31 downto 0);
    instruction_20_16, instruction_15_11        : in  std_logic_vector(4 downto 0);
    ALUSrc_out, RegDst_out                      : out std_logic;
    ALUOp_out                                   : out std_logic_vector(1 downto 0);
    Branch_out, MemRead_out, MemWrite_out       : out std_logic;
    RegWrite_out, MemToReg_out                  : out std_logic;
    PC_out                                      : out std_logic_vector(7 downto 0);
    Read_data_1_out, Read_data_2_out            : out std_logic_vector(7 downto 0);
    sign_extend_out                             : out std_logic_vector(31 downto 0);
    instruction_20_16_out, instruction_15_11_out : out std_logic_vector(4 downto 0));
```

```
----------------------------------BLOCK 3-----------------------------------------------
signal output_shift2, FA, FB: std_logic_vector(7 downto 0);
signal mux_out2: std_logic_vector(7 downto 0);
signal Operation : std_logic_vector(2 downto 0);
signal Zero                : std_logic;
signal PC_shift                                              : std_logic_vector(7 downto 0);
signal ALU                                 : std_logic_vector(7 downto 0);
signal wrb                                                   : std_logic_vector(4 downto 0);
signal Branch_out_ex, MemRead_out_ex, MemWrite_out_ex, Zero_out : std_logic;
signal RegWrite_out_ex, MemToReg_out_ex                      : std_logic;
signal PC_shift_out                                          : std_logic_vector(7 downto 0);
signal ALU_out, Read_d2_out                                  : std_logic_vector(7 downto 0);
signal wrb_out                                               : std_logic_vector(4 downto 0);
component shift_left_2 is
  generic (
    n : positive);
  port (
    input_n       : in  std_logic_vector(n downto 0);
    output_shift2 : out std_logic_vector(n+2 downto 0));
end component shift_left_2;

component EX_MEM is
  port (
    GClock, GReset                                    : in  std_logic;
    Branch, MemRead, MemWrite, Zero                   : in  std_logic;
    RegWrite, MemToReg                                : in  std_logic;
    PC                                                : in  std_logic_vector(7 downto 0);
    ALU, Read_d2                                      : in  std_logic_vector(7 downto 0);
    wrb                                               : in  std_logic_vector(4 downto 0);
    Branch_out, MemRead_out, MemWrite_out, Zero_out : out std_logic;
    RegWrite_out, MemToReg_out                        : out std_logic;
    PC_out                                            : out std_logic_vector(7 downto 0);
    ALU_out, Read_d2_out                              : out std_logic_vector(7 downto 0);
    wrb_out                                           : out std_logic_vector(4 downto 0));
end component EX_MEM;

component alu_control is
  port (
    AluOp     : in  std_logic_vector(1 downto 0);
    Funct     : in  std_logic_vector(5 downto 0);
    Operation : out std_logic_vector(2 downto 0));
end component alu_control;

component mux4n is
  generic (
    n : positive);
  port (
    i_1, i_2, i_3, i_4 : in  std_logic_vector(n downto 0);
    f                  : out std_logic_vector(n downto 0);
    s                  : in  std_logic_vector(1 downto 0));
end component mux4n;
```

```
------------------------------BLOCK 4-----------------------------------------------------
signal PCSrc               : std_logic;
signal ReadData                  : std_logic_vector(7 downto 0);
signal RegWrite_out_mem, MemToReg_out_mem : std_logic;
signal ReadData_out              : std_logic_vector(7 downto 0);
signal ALU_out_mem                  : std_logic_vector(7 downto 0);
signal wrb_out_mem                  : std_logic_vector(4 downto 0);
component data_memory is
  port (
    clock     : IN  STD_LOGIC := '1';
    data      : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
    rdaddress : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
    rden      : IN  STD_LOGIC := '1';
    wraddress : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
    wren      : IN  STD_LOGIC := '0';
    q         : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
end component data_memory;

component MEM_WB is
  port (
    GClock, GReset             : in  std_logic;
    RegWrite, MemToReg         : in  std_logic;
    ReadData                   : in  std_logic_vector(7 downto 0);
    ALU                        : in  std_logic_vector(7 downto 0);
    wrb                        : in  std_logic_vector(4 downto 0);
    RegWrite_out, MemToReg_out : out std_logic;
    ReadData_out               : out std_logic_vector(7 downto 0);
    ALU_out                    : out std_logic_vector(7 downto 0);
    wrb_out                    : out std_logic_vector(4 downto 0));
end component MEM_WB;

------------------------------BLOCK 5-----------------------------------------
signal Write_data : std_logic_vector(7 downto 0);
------------------------------Output------------------------------------------
signal other, other2: std_logic_vector(7 downto 0);
signal inst2, inst3, inst4, inst5:std_logic_vector(31 downto 0);

component mux8 is
  generic (
    n : positive);
  port (
    i_1, i_2, i_3, i_4 : in  std_logic_vector(n downto 0);
    i_5, i_6, i_7, i_8 : in  std_logic_vector(n downto 0);
    f                  : out std_logic_vector(n downto 0);
    s                  : in  std_logic_vector(2 downto 0));
end component mux8;
```

```
  ----------Hazard detection/ Forward detection unit------------

  signal MEM_WB_RegWrite, EX_MEM_RegWrite     : std_logic;
  signal ForwardA, ForwardB                   : std_logic_vector(1 downto 0);
  signal debug1                               : std_logic_vector(3 downto 0);
  signal PCWrite, IF_ID_Write, Control_mux    : std_logic;

  component Hazard_detection_unit is
    port (
      ID_EX_MemRead                      : in  std_logic;
      ID_EX_RegisterRt                   : in  std_logic_vector(4 downto 0);
      IF_ID_RegisterRs, IF_ID_RegisterRt : in  std_logic_vector(4 downto 0);
      PCWrite, IF_ID_Write, Control_mux  : out std_logic);
  end component Hazard_detection_unit;

  component Forwarding_Unit is
    port (
      EX_MEM_RegisterRd, MEM_WB_RegisterRd : in  std_logic_vector(4 downto 0);
      ID_EX_RegisterRs, ID_EX_RegisterRt   : in  std_logic_vector(4 downto 0);
      MEM_WB_RegWrite, EX_MEM_RegWrite     : in  std_logic;
      ForwardA, ForwardB                   : out std_logic_vector(1 downto 0));
  end component Forwarding_Unit;



begin  -- architecture basic
```

```
begin  -- architecture basic
  ------------------------BLOCK 1----------------------------
  enARdFF_2_1: entity work.enARdFF_2
    port map (
      i_resetBar => GReset,
      i_d        => GClock_2,
      i_enable   => '1',
      i_clock    => GClock,
      o_q        => s,
      o_qBar     => GClock_2);

  PC_1: entity work.register_nbit
    generic map (
      n => 7)
    port map (
      i_clock => GClock_2,
      load    => PCWrite,
      reset   => GReset,
      i_value => pc_input,
      o_value => pc_output);


 PC_adder: entity work.nbit_ALU
   generic map (
     n => 7)
   port map (
     a         => pc_output,
     b         => "00000001",
     Ainvert   => '0',
     Binvert   => '0',
     Operation => "10",
     result    => Alu_out1);

  instruction_memory_1: entity work.instruction_memory
    port map (
      address => pc_output,
      clock   => GClock,
      q       => instruction);
 mux_2n_1: entity work.mux_2n
    generic map (
      n => 7)
    port map (
      i_1 => Alu_out1,
      i_2 => PC_shift_out,
      sel => PCSrc,
      f   => pc_input);

  IF_ID_1: entity work.IF_ID
    port map (
      instruction     => instruction,
      pc_inc          => Alu_out1,
      GClock          => GClock_2,
      GReset          => GReset,
      IF_ID_Write     => IF_ID_Write,
      instruction_out => instruction_out,
      pc_inc_out      => pc_inc_out);
```

```
-----------------------------BLOCK 2---------------------------
register_file_1: entity work.register_file
  port map (
    GClock          => GClock,
    CReset          => CReset,
    Read_register_1 => instruction_out(25 downto 21),
    Read_register_2 => instruction_out(20 downto 16),
    Write_register  => wrb_out_mem,
    Write_data      => Write_data,
    RegWrite        => RegWrite_out_mem,
    Read_data_1     => Read_data_1_in,
    Read_data_2     => Read_data_2_in);
I_26_21<= instruction_out(25 downto 21);
I_20_16<= instruction_out(20 downto 16);
I_15_11<= wrb_out_mem;
Rd1<= Read_data_1_in;
Rd2<= Read_data_2_in;
Wr<=Write_data;
Equal_test_1: entity work.Equal_test
  port map (
    Read_data_1  => Read_data_1_in,
    Read_data_2  => Read_data_2_in,
    status_equal => status_equal);
controlpath_1: entity work.controlpath
  port map (
    Op       => instruction_out(31 downto 26),
    RegDst   => RegDst,
    ALUSrc   => ALUSrc,
    MemtoReg => MemtoReg,
    RegWrite => RegWrite,
    MemRead  => MemRead,
    MemWrite => MemWrite,
    Branch   => Branch,
    ALUOp1   => ALUOp1,
    ALUOp0   => ALUOp0,
    Jump     => Jump);

IF_Flush<=CReset and not (status_equal and Branch );
control_mux_in<= RegDst&  ALUSrc & MemtoReg & RegWrite&  MemRead &  MemWrite & Branch & ALUOp1 &  ALUOp0 & Jump;

mux_2n_5: entity work.mux_2n
  generic map (
    n => 9)
  port map (
    i_1 => control_mux_in,
    i_2 => "000000000",
    sel => Control_mux,
    f   => control_mux_out);
sign_extend_1: entity work.sign_extend
  port map (
    input_16  => instruction_out(15 downto 0),
    output_32 => sign_extend_32_in);
ALUOp<= control_mux_out(2) & control_mux_out(1);


shift_left_2_1: entity work.shift_left_2
  generic map (
    n => 5)
  port map (
    input_n       => sign_extend_32_in(5 downto 0),
    output_shift2 => output_shift2);

nbit_ALU_3: entity work.nbit_ALU
  generic map (
    n => 7)
  port map (
    a         => pc_inc_out,
    b         => instruction_out(7 downto 0),
    Ainvert   => '0',
    Binvert   => '0',
    Operation => "10",
    result    => PC_shift);

ID_EX_1: entity work.ID_EX
  port map (
    ALUSrc                 => control_mux_out(8),
    RegDst                 => control_mux_out(9),
    ALUOp                  => ALUOp,
    GClock                 => GClock_2,
    CReset                 => IF_Flush,
    Branch                 => control_mux_out(3),
    MemRead                => control_mux_out(5),
    MemWrite               => control_mux_out(4),
    RegWrite               => control_mux_out(6),
    MemToReg               => control_mux_out(7),
    PC                     => pc_inc_out,
    Read_data_1            => Read_data_1_in,
    Read_data_2            => Read_data_2_in,
    sign_extend            => sign_extend_32_in,
    instruction_20_16      => instruction(20 downto 16),
    instruction_15_11      => instruction(15 downto 11),
    ALUSrc_out             => ALUSrc_out,
    RegDst_out             => RegDst_out,
    ALUOp_out              => ALUOp_out,
    Branch_out             => Branch_out,
    MemRead_out            => MemRead_out,
    MemWrite_out           => MemWrite_out,
    RegWrite_out           => RegWrite_out,
    MemToReg_out           => MemToReg_out,
    PC_out                 => PC_out,
    Read_data_1_out        => Read_data_1_out,
    Read_data_2_out        => Read_data_2_out,
    sign_extend_out        => sign_extend_out,
    instruction_20_16_out => instruction_20_16_out,
    instruction_15_11_out => instruction_15_11_out);
```

```
-----------------------------------BLOCK 3-----------------
 alu_control_1: entity work.alu_control
   port map (
     AluOp     => AluOp_out,
     Funct     => sign_extend_out(5 downto 0),
     Operation => Operation);

 mux_2n_2: entity work.mux_2n
   generic map (
     n => 4)
   port map (
     i_1 => instruction_20_16_out,
     i_2 => instruction_15_11_out,
     sel => RegDst_out,
     f   => wrb);
 mux4n_1: entity work.mux4n
   generic map (
     n => 7)
   port map (
     i_1 => Read_data_1_out,
     i_2 => Write_data,
     i_3 => ALU_out,
     i_4 => "00000000",
     f   => FA,
     s   => ForwardA);


 mux4n_2: entity work.mux4n
   generic map (
     n => 7)
   port map (
     i_1 => Read_data_2_out,
     i_2 => ALU_out,
     i_3 => Write_data,
     i_4 => "00000000",
     f   => FB,
     s   => ForwardB);
mux_2n_3: entity work.mux_2n
   generic map (
     n => 7)
   port map (
     i_1 => FB,
     i_2 => sign_extend_out(7 downto 0),
     sel => ALUSrc_out,
     f   => mux_out2);

 nbit_ALU_2: entity work.nbit_ALU
  generic map (
    n => 7)
  port map (
    a          => FA,
    b          => mux_out2,
    Ainvert    => '0',
    Binvert    => Operation(2),
    Operation  => Operation(1 downto 0),
    result     => ALU,
    zero       => Zero);


 EX_MEM_1: entity work.EX_MEM
   port map (
     GClock       => GClock_2,
     GReset       => GReset,
     Branch       => Branch_out,
     MemRead      => MemRead_out,
     MemWrite     => MemWrite_out,
     Zero         => Zero,
     RegWrite     => RegWrite,
     MemToReg     => MemToReg,
     PC           => PC_shift,
     ALU          => ALU,
```

```
-------------------------BLOCK 4-----------------------------------
PCSrc<= Branch and (status_equal);

data_memory_1: entity work.data_memory
  port map (
    clock     => GClock,
    data      => Read_d2_out,
    rdaddress => ALU_out,
    rden      => MemRead_out_ex,
    wraddress => ALU_out,
    wren      => MemWrite_out_ex,
    q         => ReadData);

MEM_WB_1: entity work.MEM_WB
  port map (
    GClock        => GClock_2,
    GReset        => GReset,
    RegWrite      => RegWrite_out_ex,
    MemToReg      => MemToReg_out_ex,
    ReadData      => ReadData,
    ALU           => ALU_out,
    wrb           => wrb_out,
    RegWrite_out  => RegWrite_out_mem,
    MemToReg_out  => MemToReg_out_mem,
    ReadData_out  => ReadData_out,
    ALU_out       => ALU_out_mem,
    wrb_out       => wrb_out_mem);
```

```
----------------------BLOCK 5-----------------------------

  mux_2n_4: entity work.mux_2n
    generic map (
      n => 7)
    port map (
      i_1 => ReadData_out,
      i_2 => ALU_out_mem,
      sel => MemToReg_out_mem,
      f   => Write_data);

--------------------Output----------------------------

    other<= '0'& RegDst& Jump& MemRead & MemtoReg & ALUOp & ALUSrc;
  other2<=Alu_out1;
 mux8_1: entity work.mux8
   generic map (
     n => 7)
   port map (
     i_1 => pc_output,
     i_2 => ALU,
     i_3 => Read_data_1_in,
     i_4 => Read_data_2_in,
     i_5 => Write_data,
     i_6 => other,
     i_7 => other2,
     i_8 => pc_input,
     f   => MuxOut,
     s   => ValueSelect);

          BranchOut<= Branch;
          ZeroOut<= Zero;
          MemWriteOut<= MemRead_out_ex;
          RegWriteOut<= RegWrite_out_mem;



  instruction_1: entity work.register_nbit
    generic map (
      n => 31)
    port map (
      i_clock => GClock_2,
      load    => '1',
      reset   => GReset,
      i_value => instruction_out,
      o_value => inst2);
   instruction_2: entity work.register_nbit
    generic map (
      n => 31)
    port map (
      i_clock => GClock,
      load    => '1',
      reset   => GReset,
      i_value => inst2,
      o_value => inst3);
   instruction_3: entity work.register_nbit
    generic map (
      n => 31)
    port map (
      i_clock => GClock_2,
      load    => '1',
      reset   => GReset,
      i_value => inst3,
      o_value => inst4);
   instruction_4: entity work.register_nbit
    generic map (
```

```
mux8_2: entity work.mux8
  generic map (
    n => 31)
  port map (
    i_1 => instruction,
    i_2 => instruction_out,
    i_3 => inst2,
    i_4 => inst3,
    i_5 => inst4,
    i_6 => inst5,
    i_7 => "00000000000000000000000000000000",
    i_8 => "00000000000000000000000000000000",
    f   => InstructionOut,
    s   => InstrSelect);


----------Hazard detection/ Forwarding unit------------------

Forwarding_Unit_1: entity work.Forwarding_Unit
  port map (
    EX_MEM_RegisterRd => wrb_out,
    MEM_WB_RegisterRd => wrb_out_mem,
    ID_EX_RegisterRs  => instruction_25_21_out,
    ID_EX_RegisterRt  => instruction_20_16_out,
    MEM_WB_RegWrite   => RegWrite_out_mem,
    EX_MEM_RegWrite   => RegWrite_out_ex,
    ForwardA          => ForwardA,
    ForwardB          => ForwardB);


Hazard_detection_unit_1: entity work.Hazard_detection_unit
  port map (
    ID_EX_MemRead    => MemRead_out,
    ID_EX_RegisterRt => instruction_20_16_out,
    IF_ID_RegisterRs => instruction_25_21_out,
    IF_ID_RegisterRt => instruction_out(20 downto 16),
    PCWrite          => PCWrite,
    IF_ID_Write      => IF_ID_Write,
    Control_mux      => Control_mux);

debug1<=PCWrite & IF_ID_Write & ALUSrc_out & IF_Flush;

debug: entity work.mux8
  generic map (
    n => 3)
  port map (
    i_1 => debug1,
    i_2 => "0000",
    i_3 => "0000",
    i_4 => "0000",
    i_5 => "0000",
    i_6 => "0000",
    i_7 => "0000",
    i_8 => "0000",
    f   => DebugOut,
    s   => DebugSelect);

end architecture basic;
```

Figure13: Code complet de l'entité finale pipelinedproc

Ceci est le code code complet de l'entité finale implémentant un processeur à pipeline à l'aide de tous les composants vus précédemment.

III-Realisiation Reelle

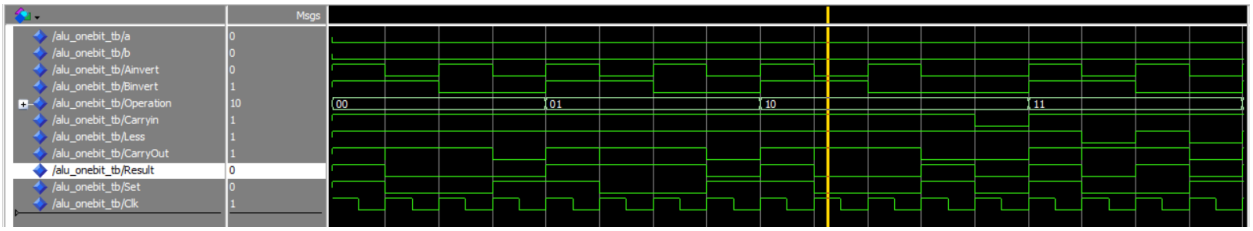


Figure14 : Waveforme de alu_onebit

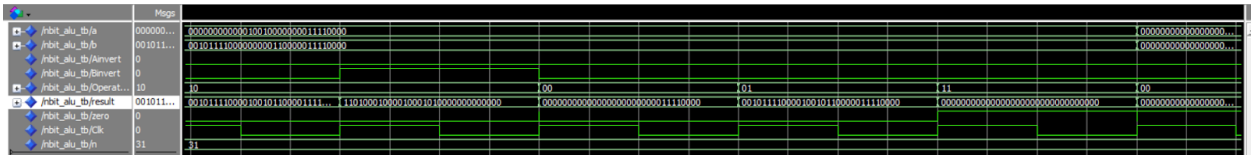


Figure15 : Waveforme de nbit_alu

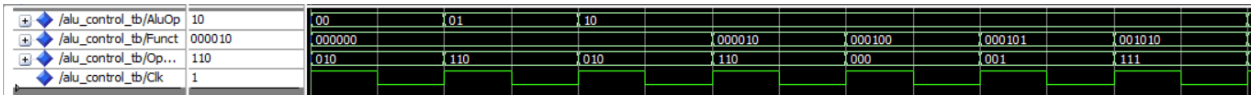


Figure16 : Waveforme de alu_control

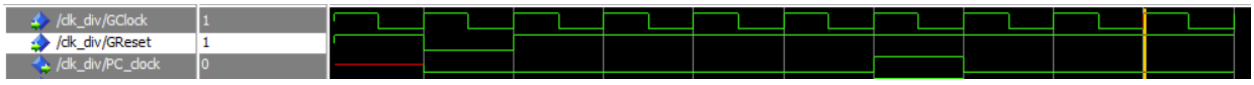


Figure16: Waveforme de clk_div

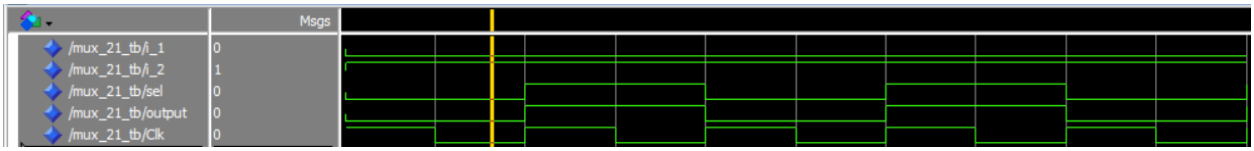


Figure17: Waveforme de mux_21

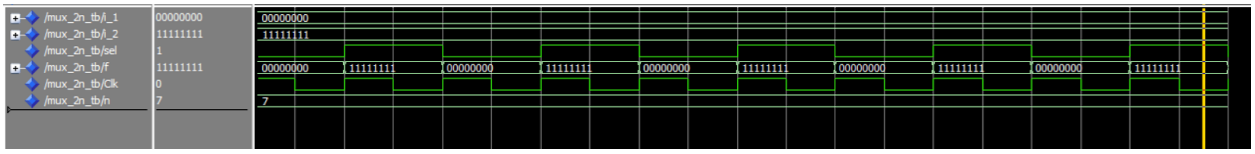


Figure18 : Waveforme de mux_2n

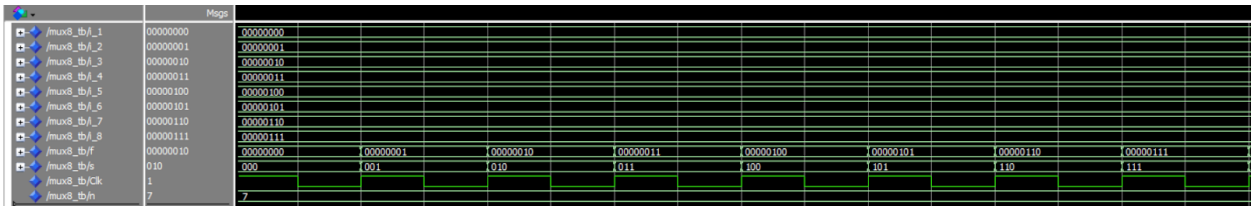


Figure19 : Waveforme de mux8

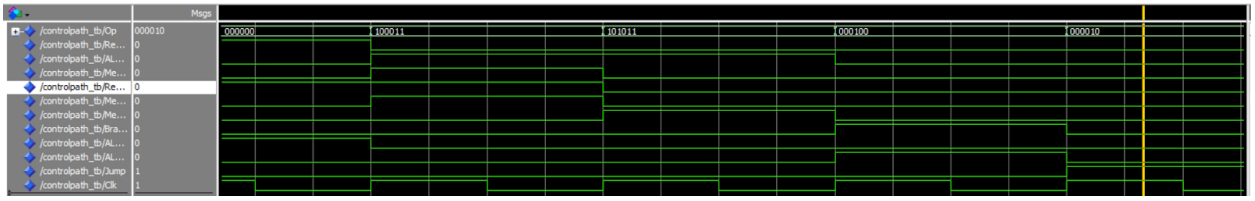


Figure20 : Wavforme de controlpath

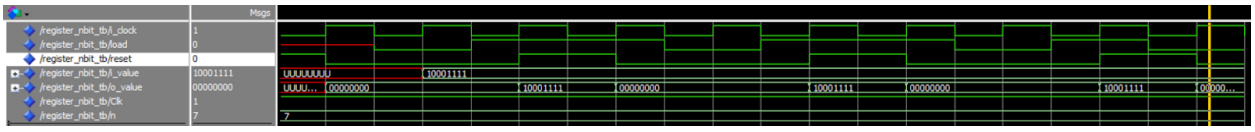


Figure21 : Waveforme de register_nbit

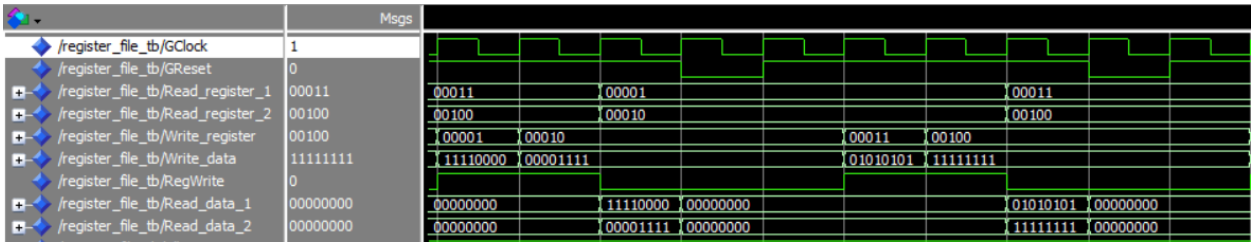


Figure22: Waveforme de register_file

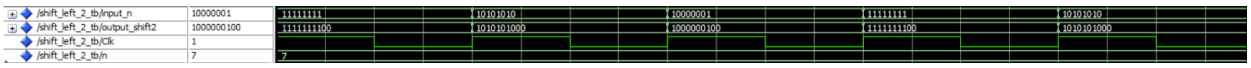


Figure23 : Waveform de shift_left_2

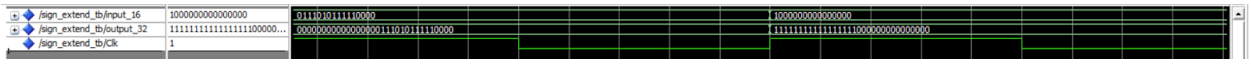


Figure24 : Waveforme de sign_extend

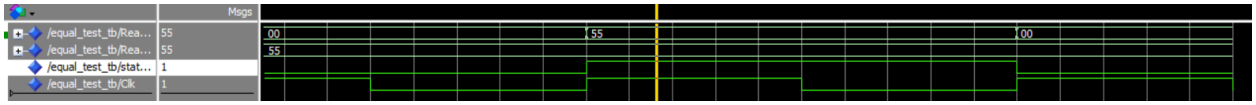


Figure : Waveforme de Equal test

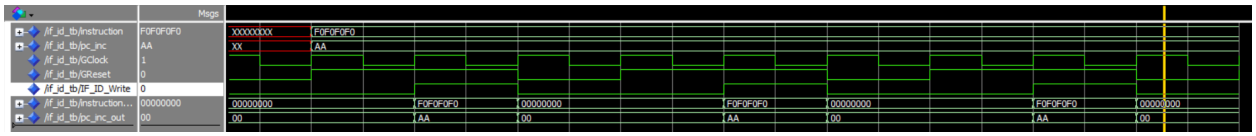


Figure : Waveforme de if_id

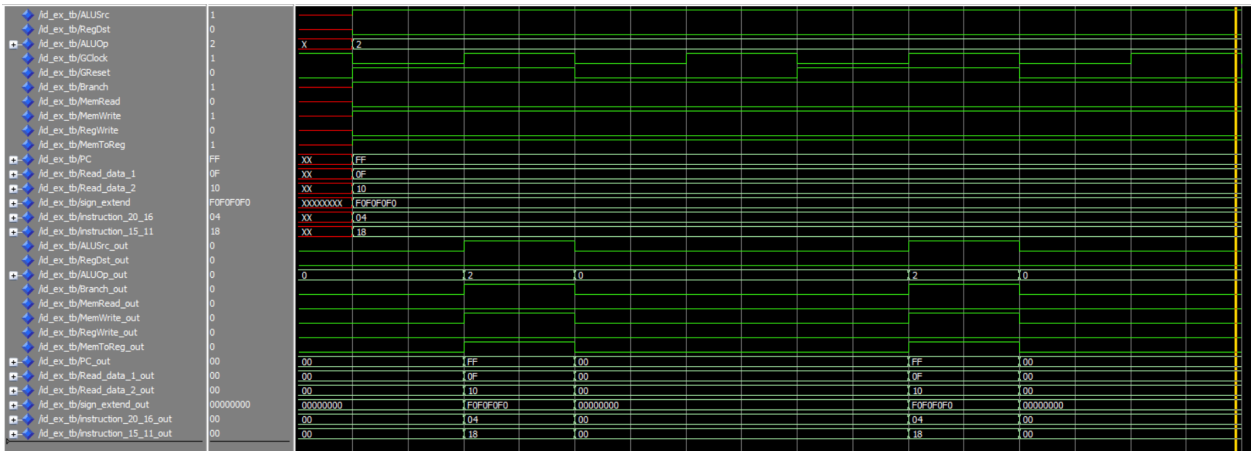


Figure : Waveforme de id_ex

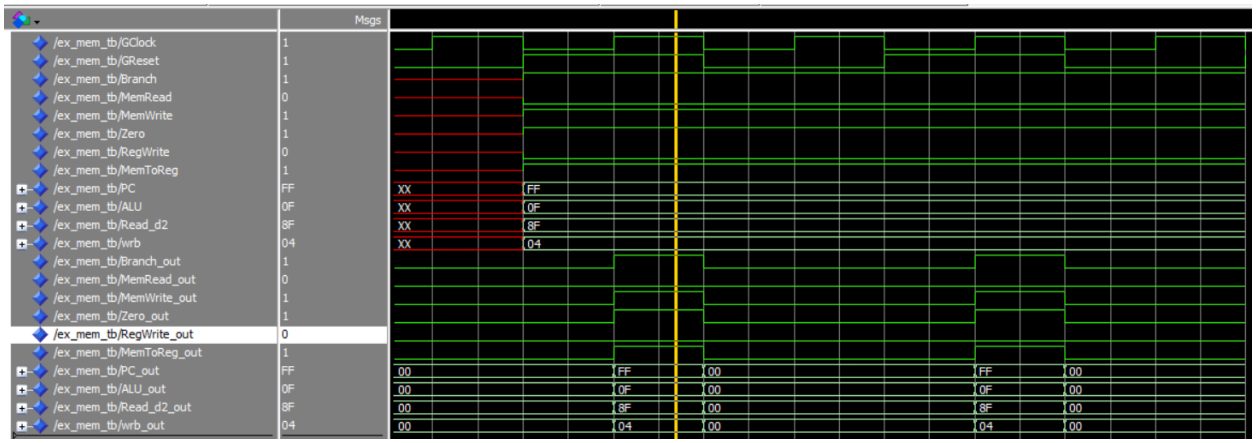


Figure : Waveforme de ex_mem

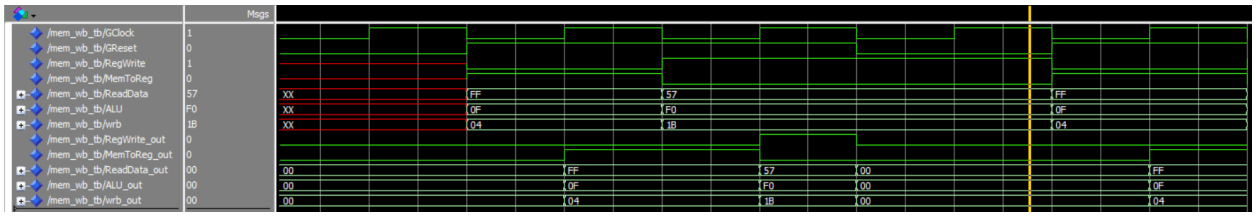


Figure : Waveforme de mem_wb

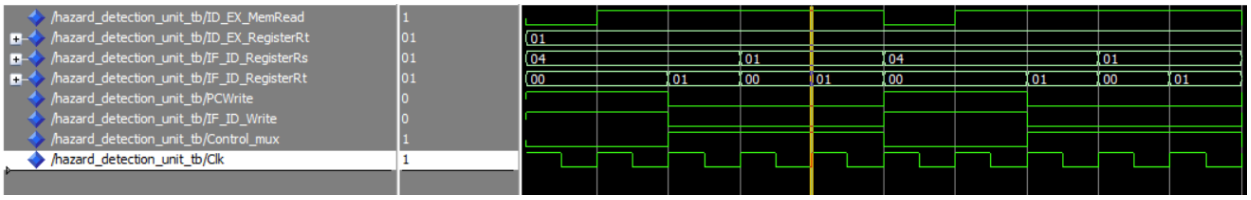


Figure : Waveforme de Hazard_detection_unit

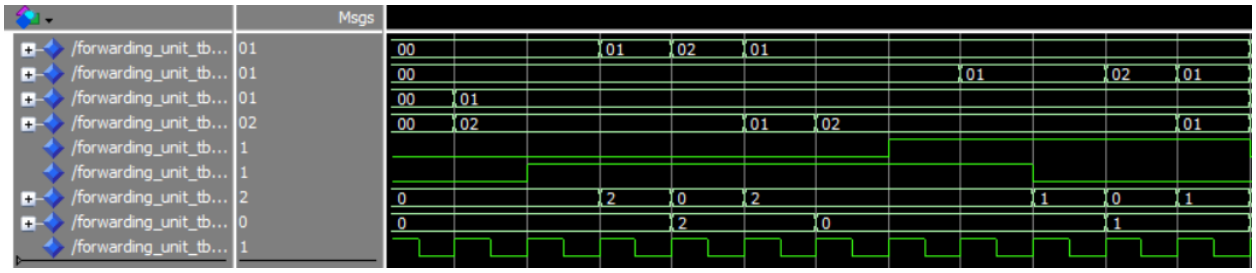


Figure: Waveforme de Forwarding_unit

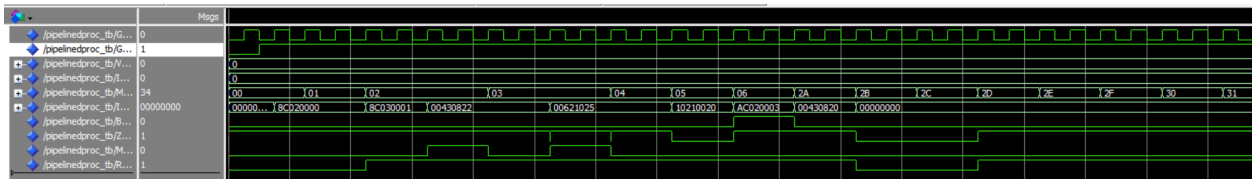


Figure : Waveforme de pipelinedproc

Bonus : DÉMO sur la carte Altera.

Performance

Performance processeur à cycle simple :
delai_alu= 8bits x 2 delai porte logique x 0.01= 0.16 ns

Delai processeur = 0.2ns (instruction mem) + 0.1 ns (reg file) + 0.2 (data memory) + 0.16 ALU + 0.1 ns (write back)= 0.76
Avec 1 cycle = 0.2ns

Performance processeur à cycle pipeline

La performance de la première instruction est la même, mais lorsque le nombre d'instruction qu'on exécute augmente on s'approche à une performance de une instruction par cycle.

Délai processeur pipeline: 0.2ns

Les hasards de contrôles diminuent la performance.

Discussion et Conclusion

Ce laboratoire a été assez compliqué comparé aux précédents. Tout d'abord, nous n'avons pas eu besoin de rajouter plusieurs composantes vu que le processeur Pipeline se fait très bien à partir du processeur à cycle simple implémenté au lab 2. De plus, la conception graphique du Datapath été beaucoup plus compliquée pour ce laboratoire dû à la grande quantité de composantes que nous avions utilisé. Ainsi, le testing a été très laborieux et le débogage notamment dû à des problèmes de synchronisation ont pris une grande partie du temps. Il faut qu'on adapte nos méthodes de testing pour les grands projets Cependant, les objectifs du laboratoire, concevoir, réaliser et de tester un processeur pipeline RISC a été atteint. On a approfondi la théorie vue en classe, tout en appliquant les leçons de conception apprises dans le laboratoire précédent, notamment au niveau du testing.