Casa abierta al tiempo

ARQUITECTURA MIPS EN VHDL

Arquitectura de Computadoras

Esaú Acosta Lara

2153036850

INTRODUCCIÓN

John L. Hennessy, junto con su equipo, en la Universidad de Stanford comenzó a trabajar en lo que es el primer procesador MIPS. El concepto básico era para aumentar el rendimiento mediante el uso de tuberías de instrucciones profundas. Canalización como "la ciencia básica" era muy conocida antes, pero no se convirtió en su potencial máximo. Las CPU se construyen a partir de una serie de subunidades dedicadas como decodificadores de instrucciones, ALU, unidades de carga / almacenamiento, etc. En un diseño no optimizado tradicional, una instrucción particular en una secuencia del programa debe estar listo antes de la próxima puede ser emitida para su ejecución, en una arquitectura segmentada, las instrucciones sucesivas son posibles suponer en la ejecución.

Un aspecto importante del diseño de MIPS era para adaptarse a cada sub-fase, cuentos como memoria caché de acceso, de todas las instrucciones en un ciclo, eliminación de esta herramienta para cualquier necesidad de enclavamiento, y que permitía un solo ciclo de rendimiento.

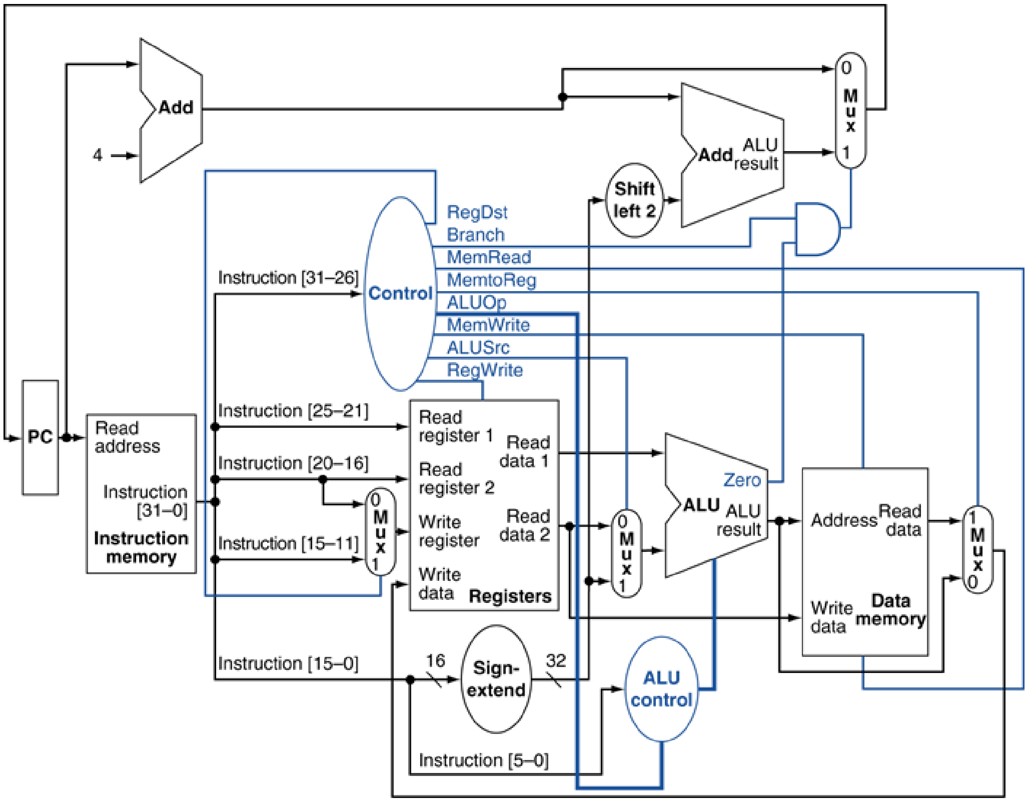
En otros aspectos el diseño MIPS fue en gran medida un típico diseño RISC. Para guardar los bits de la palabra de instrucción, diseños RISC reducir el número de instrucciones para codificar. El diseño MIPS usa 6 bits de la palabra de 32 bits para el código de operación básico, y el resto puede contener una dirección de salto solo 26 bits o puede tener hasta cuatro campos de 5 bits que especifica hasta tres registros además de un valor de cambio combinado con otros 6 bits de código de operación; Otro formato, entre varios, específico dos registros combinados con un valor inmediato de 16 bits, etc. Esto permite que esta CPU para cargar hasta la instrucción y los datos que se necesitan en un solo ciclo, mientras que no lo sea RISC, tales como el MOS Technology 6502, por ejemplo, requiere ciclos separados para cargar el código de operación y los datos. Esta fue una de las mejoras de rendimiento importante que RISC ofrece. Sin embargo, los diseños modernos de no-RISC logran esta velocidad por otros medios

MIPS (microprocesador sin etapas de interconexión de tuberías; microprocesador sin bloqueos en las etapas de segmentación) es una arquitectura diseñada para optimizar la segmentación en unidades de control y para facilitar la generación automática de código máquina por parte de los compiladores. Existen múltiples revisiones del conjunto de instrucciones MIPS:

Versión	Ancho de los registros	Procesadores	Comentarios
MIPS I	32	R2000, R3000	Versión comercial del procesador MIPS de la Universidad de Stanford
MIPS II	32	R6000	Base del estándar MIPS32
MIPS III	64	R4000	Primera arquitectura MIPS de 64 bits
MIPS IV	64	R5000, R10000	Actualización menor de MIPS III
MIPS V	64	-	Base del estándar MIPS64

Implementaciones MIPS se usa principalmente en sistemas embebidos, cuentos como dispositivos de Windows CE, enrutadores, puertas de enlace y consolas de videojuegos como Sony PlayStation 2 y PlayStation Portable. Hasta finales de 2006, también se usa en muchos de los productos informáticos de SGI. Implementaciones MIPS también se utilizaron por Digital Equipment Corporation, NEC, Pyramid Technology, Siemens Nixdorf, Tandem Computers y otros durante los años 1980 y 1990. Entre mediados y finales de 1990, se estima que uno de cada tres microprocesadores RISC fue fue una implementación MIPS.

ESTRUCTURA



INSTRUCCIONES

- Tipo R
(shamt: shift amount en instrucciones de desplazamiento)

Cod. Op.	Registro fuente 1	Registro fuente 2	Registro destino		Funct
xxxxxx	rs	rt	rd	shamt	funct
6	5	5	5	5	6
31-26	25-21	20-16	15-11	10-6	5-0

- Tipo I
(carga o almacenamiento, ramificación condicional, operaciones con inmediatos)

Cod. Op.	Registro base	Registro destino	Registro destino	Desplazamiento
xxxxxx	rs	rt	rd	Offset
6	5	5	5	16
31-26	25-21	20-16	15-11	15-0

- Tipo J
(salto incondicional)

Cod. Op.	Dirección destino
xxxxxx	target
6	26
31-26	20-16

REGISTROS

Nombre Registro	Número	Uso
zero	0	Constante 0
at	1	Reservado para el assembler
v0	2	Para evaluación de expresiones y retorno de resultados de la función
v1	3	
a0	4	Argumento1
a1	5	Argumento 2
a2	6	Argumento 3
a3	7	Argumento 4
t0	8	Temporal (no se preserva a través de los llamados)
t1	9	Temporal (no se preserva a través de los llamados)

t2	10	Temporal (no se preserva a través de los llamados)
t3	11	Temporal (no se preserva a través de los llamados)
t4	12	Temporal (no se preserva a través de los llamados)
t5	13	Temporal (no se preserva a través de los llamados)
t6	14	Temporal (no se preserva a través de los llamados)
t7	15	Temporal (no se preserva a través de los llamados)
s0	16	Temporal que debe preservarse entre llamados a funciones
s1	17	Temporal que debe preservarse entre llamados a funciones
s2	18	Temporal que debe preservarse entre llamados a funciones
s3	19	Temporal que debe preservarse entre llamados a funciones
s4	20	Temporal que debe preservarse entre llamados a funciones
s5	21	Temporal que debe preservarse entre llamados a funciones
s6	22	Temporal que debe preservarse entre llamados a funciones
s7	23	Temporal que debe preservarse entre llamados a funciones
t8	24	Temporal (no se preserva a través de los llamados)
t9	25	Temporal (no se preserva a través de los llamados)
k0	26	Reservado para el núcleo del sistema operativo
k1	27	Reservado para el núcleo del sistema operativo
gp	28	Puntero al área global de datos
sp	29	Puntero al tope de la pila. Stack pointer
fp	30	Puntero a zona de variables en la pila. Frame pointer
ra	31	Dirección de retorno (usado en invocaciones a funciones)

DESARROLLO

Control_alu.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity control_alu is
    port(
        ALUOp: in std_logic_vector(1
downto 0);
        FuncCode: in std_logic_vector(5
downto 0);
        ALUCtl: out std_logic_vector(3
downto 0)
    );
end control_alu;

architecture estructural of control_alu is
    begin
    ALUCtl <= "0010" when ALUOp(1) =
'0' and ALUOp(0) = '0' else
    "0110" when ALUOp(1) = '0' and
ALUOp(0) = '1' else
    "0010" when ALUOp(1) = '1' and
ALUOp(0) = '0' and FuncCode(3) = '0'
and FuncCode(2) = '0' and
 FuncCode(1) = '0' and FuncCode(0) =
'0' else
    "0110" when ALUOp(1) = '1' and
FuncCode(3) = '0' and FuncCode(2) =
'0' and FuncCode(1) = '1' and
FuncCode(0) = '0' else
    "0000" when ALUOp(1) = '1' and
ALUOp(0) = '0' and FuncCode(3) = '0'
and FuncCode(2) = '1' and
FuncCode(1) = '0' and FuncCode(0) =
'0' else
    "0001" when ALUOp(1) = '1' and
ALUOp(0) = '0' and FuncCode(3) = '0'
and FuncCode(2) = '1' and
FuncCode(1) = '0' and FuncCode(0) =
'1' else
    "0111" when ALUOp(1) = '1' and
FuncCode(3) = '1' and FuncCode(2) =
'0' and FuncCode(1) = '1' and
FuncCode(0) = '0';

end estructural;
```

Control_unit.vhd

```
--Unidad de control MIPS
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity control_unit is
    port(
        op : in std_logic_vector(5 downto
0);
        RegDst:out std_logic;
        ALUSrc:out std_logic;
downto 0);
    begin
        ctrl_word <= "100100010" when
op = "000000" else --instrucciones
Type-R
                    "011110000" when op =
"100011" else --lw  :0x23
                    "X1X001000" when op =
"101011" else --sw  :0x2B
```

```
        MemToReg:out std_logic;
        RegWrite : out std_logic;
        MemRead, MemWrite, Branch :
out std_logic;
        ALUOp : out std_logic_vector(1
downto 0)
    );
end control_unit;

architecture Estructura of control_unit
is
    signal ctrl_word: std_logic_vector(8
```

```
                    "X0X000101" when op =
"000100" else --beq :0x04
                    "000000000";
        RegDst <= ctrl_word(8);
        ALUSrc <= ctrl_word(7);
        MemToReg <= ctrl_word(6);
        RegWrite <= ctrl_word(5);
        MemRead <= ctrl_word(4);
        MemWrite <= ctrl_word(3);
        Branch <= ctrl_word(2);
        ALUOp <= ctrl_word(1 downto 0);

end Estructura;
```

Register_File.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
entity Register_File is
    port (RegWrite:in std_logic; --señal
de control
    RR1: in std_logic_vector (4 downto
0); --numero de registro 1
    RR2: in std_logic_vector (4 downto
0); --numero de registro 2
    Write_Register: in std_logic_vector
(4 downto 0);
    Write_Data: in std_logic_vector (31
downto 0); --En el anterior numero
guardar los 8 bytes de esta entrada
    RD1: out std_logic_vector (31
downto 0); --Salida de 32 bits.
    RD2: out std_logic_vector (31
downto 0)); --Salida de 32 bits.
end Register_File;

architecture Behavioral of
Register_File is
    TYPE memoria IS ARRAY (0 TO
31) OF std_logic_vector(31 downto 0);
    signal data_mem:memoria:=(
    x"00000000",
```

```
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000"
    );
begin

RD1<=data_mem(to_integer(unsigned(
RR1)));
```

```
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
    x"00000000",
```

```

RD2<=data_mem(to_integer(unsigned(
RR2)));

process(RegWrite)
    begin
        if (RegWrite='1') then

data_mem(to_integer(unsigned(Write_
Register)))<=Write_Data;
        end if;
    end process;

end Behavioral;
```

Registrer_File_v2.vhd

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

entity Register_File is
    port (
        clk: in std_logic;
        reset: in std_logic;
        RegWrite:in std_logic; --señal de
control
        RR1: in std_logic_vector (4
downto 0); --numero de registro 1
        RR2: in std_logic_vector (4
downto 0); --numero de registro 2
        Write_Register: in
std_logic_vector (4 downto 0);
        Write_Data: in std_logic_vector
(31 downto 0); --En el anterior numero
guardar los 8 bytes de esta entrada
        RD1: out std_logic_vector (31
downto 0); --Salida de 32 bits.
        RD2: out std_logic_vector (31
downto 0)); --Salida de 32 bits
  end Register_File;
```

```
std_logic_vector(31 downto 0);
        reset: in std_logic
    );
    end component re_hab_buff;
  end structural ;

    signal decoderRW_d_data:
std_logic_vector(31 downto 0);
    signal decoderA_d_data:
std_logic_vector(31 downto 0);
    signal decoderB_d_data:
std_logic_vector(31 downto 0);

  begin
    decoderRW : dec5a32
      port map(
        a => Write_Register,
        d => decoderRW_d_data,
        e => RegWrite
      );
    decoderA : dec5a32
      port map(
        a => RR1,
        d => decoderA_d_data,
        e => '1'
  architecture structural of Register_File is

    component dec5a32
      port (
        a: in std_logic_vector(4 downto 0);
        e: in std_logic;
        d: out std_logic_vector(31 downto 0)
      ) ;
    end component dec5a32;

    component re_hab_buff
    port(
        data_in: in std_logic_vector(31 downto 0);
        eBusA: in std_logic;
        eBusB: in std_logic;
        e: in std_logic;
        clk: in std_logic;
        dataBusA_out: out std_logic_vector(31 downto 0);
        dataBusB_out: out
```

```
      );
    decoderB : dec5a32
      port map(
        a => RR2,
        d => decoderB_d_data,
        e => '1'
      );

    alus : for i in 0 to 31 generate

        regis0_31 :  re_hab_buff port map(
            data_in => Write_Data,
            eBusA => decoderA_d_data(i),
            eBusB => decoderB_d_data(i),
            e => decoderRW_d_data(i),
            clk => clk,
            dataBusA_out => RD1,
            dataBusB_out => RD2,
            reset => reset
        );

    end generate ;
```

Data_memory.vhd

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

entity data_memory is
  port(
    clk, MemRead, MemWrite : in std_logic;
    address : in  std_logic_vector(31 downto 0);
    datain  : in  std_logic_vector(31 downto 0);
    dataout : out std_logic_vector(31 downto 0)
  );
end data_memory;
```

```
          memDatos(to_integer(unsigned(address))+2) <= datain(15 downto 8);

          memDatos(to_integer(unsigned(address))+3) <= datain(7 downto 0);
                    end if;
               end if;
  end process;

  --Asynchronous reading
  dataout(31 downto 24) <= memDatos(to_integer(unsigned(address)))
                        when MemRead = '1' else (others => 'Z');
```

```

architecture behavioral of
data_memory is
        type tipoRAM is array(0 to 255)
of std_logic_vector(7 downto 0);
  signal memDatos : tipoRAM;
begin
        process(clk)
  begin
                if rising_edge(clk) then
                        if MemWrite = '1'
then
        memDatos(to_integer(unsigned(
address)))   <= datain(31 downto 24);

        memDatos(to_integer(unsigned(
address))+1) <= datain(23 downto 16);
```

```
  dataout(23 downto 16) <=
memDatos(to_integer(unsigned(addres
s))+1)
                    when MemRead = '1'
else (others => 'Z');
  dataout(15 downto 8)  <=
memDatos(to_integer(unsigned(addres
s))+2)
                    when MemRead = '1'
else (others => 'Z');
  dataout(7 downto 0)   <=
memDatos(to_integer(unsigned(addres
s))+3)
                    when MemRead = '1'
else (others => 'Z');

end behavioral;
```

Instruction_memory.vhd

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

entity instruction_memory is
  port (
    addr: in std_logic_vector(31 downto
0);
    instruction: out std_logic_vector(31
downto 0)
  ) ;
end instruction_memory;

architecture structural of
instruction_memory is

  component dec5a32
    port (
      a: in std_logic_vector(4 downto 0);
      e: in std_logic;
      d: out std_logic_vector(31 downto
0)
    );
```

```
dec_line(7) or dec_line(11) or
                dec_line(12) or
dec_line(16);
  instruction(16) <= dec_line(0) or
dec_line(2) or
                dec_line(3) or dec_line(5)
or dec_line(7) or
                dec_line(8) or dec_line(9)
or dec_line(13) or dec_line(16);
  instruction(15) <= dec_line(2) or
dec_line(10) or dec_line(11) or
                dec_line(15);
  instruction(14) <= dec_line(3) or
dec_line(5) or dec_line(7) or
dec_line(8) or
                dec_line(10) or
dec_line(12) or dec_line(13) or
                dec_line(15);
  instruction(13) <= dec_line(10) or
dec_line(15);
  instruction(12) <= dec_line(2) or
dec_line(7) or dec_line(10) or
```

```
  end component dec5a32;

  signal dec_line: std_logic_vector(31
downto 0);

begin

  decoder5x32_u: dec5a32 port map(
    a => addr(6 downto 2),
    d => dec_line,
    e=> '1'
  );

--TODO: Make the generation of
decoder lines automatically

  instruction(31) <= dec_line(0) or
dec_line(1) or dec_line(16);
  instruction(30) <= '0';
  instruction(29) <= dec_line(16);
  instruction(28) <= dec_line(4) or
dec_line(6) or dec_line(10) or
                     dec_line(15);
  instruction(27) <= dec_line(0) or
dec_line(1) or dec_line(16);
  instruction(26) <= dec_line(0) or
dec_line(1) or dec_line(16);
  instruction(25) <= dec_line(14);
  instruction(24) <= dec_line(7) or
dec_line(8) or dec_line(9) or
                     dec_line(13);
  instruction(23) <= '0';
  instruction(22) <= dec_line(7) or
dec_line(14);
  instruction(21) <= dec_line(8) or
dec_line(10) or dec_line(15);

  instruction(20) <= dec_line(0) or
dec_line(1) or dec_line(2) or
                     dec_line(3) or dec_line(5)
or dec_line(7) or dec_line(8) or
dec_line(13) or dec_line(16);
  instruction(19) <= dec_line(9) or
dec_line(11) or dec_line(12) or
                     dec_line(14);
                     dec_line(11) or
dec_line(12) or dec_line(15);
  instruction(11) <= dec_line(2) or
dec_line(5) or dec_line(8) or
                     dec_line(9) or
dec_line(10) or dec_line(11) or
                     dec_line(14) or
dec_line(15);
  instruction(10) <= dec_line(10) or
dec_line(15);
  instruction( 9) <= dec_line(10) or
dec_line(15);
  instruction( 8) <= dec_line(10) or
dec_line(15);
  instruction( 7) <= dec_line(10) or
dec_line(15);
  instruction( 6) <= dec_line(10) or
dec_line(15);
  instruction( 5) <= dec_line(2) or
dec_line(3) or dec_line(5) or
                     dec_line(7) or dec_line(8)
or dec_line(9) or
                     dec_line(10) or
dec_line(11) or dec_line(12) or
                     dec_line(13) or
dec_line(14) or dec_line(15);
  instruction( 4) <= dec_line(10) or
dec_line(15);
  instruction( 3) <= dec_line(4) or
dec_line(9) or
                     dec_line(10) or
dec_line(14) or dec_line(16);
  instruction( 2) <= dec_line(1) or
dec_line(2) or dec_line(3) or
                     dec_line(5) or
dec_line(10) or dec_line(11) or
                     dec_line(12) or
dec_line(15);
  instruction( 1) <= dec_line(6) or
dec_line(9) or dec_line(14);
  instruction( 0) <= dec_line(2) or
dec_line(3) or dec_line(4) or
dec_line(5) or
                     dec_line(11) or
dec_line(15);
```

```
  instruction(18) <= '0';
  instruction(17) <= dec_line(1) or
```

```
end structural ; -- structural
```

alu32bits.vhd

```
   library IEEE;
use IEEE.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity alu32bits is port(
   entrada1:in std_logic_vector(31
downto 0);
   entrada2:in std_logic_vector(31
downto 0);
   control:in std_logic_vector (3 downto
0);
   result: out std_logic_vector(31
downto 0);
   zero:out std_logic
);

end alu32bits;

architecture behavioral of alu32bits is
   begin

result<=std_logic_vector(signed(entrad
a1)+signed(entrada2))when(control="0
010")else

std_logic_vector(signed(entrada1)-
signed(entrada2))when(control="0110"
)
```

```
else
      (entrada1) and
(entrada2)when(control="0000")else
      (entrada1) or
(entrada2)when(control="0001")else

"00000000000000000000000000000
01" when (control="0111" and
entrada1<entrada2) else

"00000000000000000000000000000
00" when (control="0111");

      zero<= '1' when
(std_logic_vector(signed(entrada1)) =
std_logic_vector(signed(entrada2)) and
control="0110")else
         '1' when
(std_logic_vector(signed(entrada1)) =
std_logic_vector(-signed(entrada2))
and control="0010")else
         '1' when (control="0110")else
         '0';



   end behavioral;
```

Processor.vhd

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

```
   signal dtadm:std_logic_vector(31
downto 0);
```

```
entity processor is
    port (
        clock: in std_logic;
        reset: in std_logic
    ) ;
end processor;

architecture structural of processor is

  component datapath is
    port (
      data_in: in std_logic_vector(31
downto 0);
      instruction: in std_logic_vector(25
downto 0);
      --RegDst, ALUSrc, MemToReg,
RegWrite, PCSrc
      ctrlword: in std_logic_vector(4
downto 0);
      alu_code: in std_logic_vector(3
downto 0);
      data_out, mem_addr: out
std_logic_vector(31 downto 0);
      instr_addr: out std_logic_vector(31
downto 0);
      clock, reset: in std_logic
    );
  end component datapath;

  component instruction_memory
    port (
      addr: in std_logic_vector(31
downto 0);
      instruction: out std_logic_vector(31
downto 0)
    ) ;
  end component instruction_memory;

  component data_memory
    port(
      clk, MemRead, MemWrite : in
std_logic;
      address : in  std_logic_vector(31
downto 0);
```

```
    signal dpadm:std_logic_vector(31
downto 0);
    signal cablememread:std_logic;
    signal cablememwrite:std_logic;
    signal aluopc:std_logic_vector (1
downto 0);
    --signal MemRead, MemWrite:
std_logic;
    --signal initial_data, data,
initial_addr, address:
std_logic_vector(31 downto 0);
    --signal clock: std_logic := '0';
    --signal reset: std_logic := '1';
    --signal finish_flag: boolean := false;

    --signal initial_complete: boolean :=
false;
    --signal instruction_addr, instruction:
std_logic_vector(31 downto 0);
    --RegDst, ALUSrc, MemToReg,
RegWrite, MemRead, MemWrite,
PCSrc, ALUOp1 y ALUOp0

    begin


 --MemWrite <= '1' when
initial_complete = false else
unidadctrl_dataMemo_MemWrite;
 --MemRead  <= '0' when
initial_complete = false else
unidadctrl_dataMemo_MemRead;
 -- data <= initial_data when
initial_complete = false else
ruta_dataMemo_data_out;
 -- address <= initial_addr when
initial_complete = false else
ruta_dataMemo_addresRes;
 memoriainstrucciones:
instruction_memory
      port map(
        addr => direccion1,
        instruction => instruccion
      );
     unioncondatapath : datapath
```

```
      datain  : in  std_logic_vector(31
downto 0);
      dataout : out std_logic_vector(31
downto 0)
  );
  end component data_memory;

  component control_unit
  port (
    op : in std_logic_vector(5 downto 0);
    RegDst:out std_logic;
    ALUSrc:out std_logic;
    MemToReg:out std_logic;
    RegWrite : out std_logic;
    MemRead, MemWrite, Branch : out
std_logic;
    ALUOp : out std_logic_vector(1
downto 0)
  );
  end component control_unit;

  component control_alu
    port (
      ALUOp: in std_logic_vector(1
downto 0);
        FuncCode: in std_logic_vector(5
downto 0);
        ALUCtl: out std_logic_vector(3
downto 0)
    );
  end component control_alu;

    signal
salidamemo:std_logic_vector(31
downto 0);
    signal
instruccion:std_logic_vector(31 downto
0);
    signal
palabracontrol:std_logic_vector(4
downto 0);
    signal
salidaalucontrol:std_logic_vector(3
downto 0);
```

```
      port map(
        clock => clock,
        reset => reset,
        data_in => salidamemo,
        instruction =>instruccion(25
downto 0),
        ctrlword => palabracontrol,
        alu_code => salidaalucontrol,
        mem_addr => dtadm,
        data_out => dpadm,
        instr_addr => direccion1
      );

    memoriadedatos : data_memory
      port map(
        clk => clock,
        --MemRead => MemRead,
        MemRead => cablememread,
        --MemWrite => MemWrite,
        MemWrite => cablememwrite,
        --address => address,
        address => dtadm,
        --datain => data,
        datain => dpadm,
        dataout => salidamemo
      );
    unidaddecontrol : control_unit
      port map(
        op => instruccion(31 downto 26),
        RegDst => palabracontrol(4),
        ALUSrc => palabracontrol(3),
        MemToReg => palabracontrol(2),
        RegWrite => palabracontrol(1),
        MemRead => cablememread,
        MemWrite => cablememwrite,
        Branch => palabracontrol(0),
        ALUOp => aluopc(1 downto 0)
      );
    alucontrol : control_alu
      port map(
        ALUOp => aluopc,
        FuncCode => instruccion(5
downto 0),
        ALUCtl => salidaalucontrol
      );
    signal direccion1:std_logic_vector(31 downto 0);

  end structural ; -- structural
```

datapath.vhd

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

entity datapath is
    port (
        data_in: in std_logic_vector(31 downto 0);
        instruction: in std_logic_vector(25 downto 0);
        --RegDst, ALUSrc, MemToReg, RegWrite, Branch
        ctrlword: in std_logic_vector(4 downto 0);
        alu_code: in std_logic_vector(3 downto 0);
        data_out, mem_addr: out std_logic_vector(31 downto 0);
        instr_addr: out std_logic_vector(31 downto 0);
        clock, reset: in std_logic
    ) ;
  end datapath;

  architecture structural of datapath is

    component Register_File
      port (
        clk: in std_logic;
        reset: in std_logic;
        RegWrite:in std_logic; --señal de control
        RR1: in std_logic_vector (4 downto 0); --numero de registro 1
        RR2: in std_logic_vector (4 downto 0); --numero de registro 2
        Write_Register: in std_logic_vector (4 downto 0);
    signal salida2r:std_logic_vector(31 downto 0);
    signal se_a_sll2:std_logic_vector(31 downto 0);
    signal sll2_a_ADDALU:std_logic_vector(31 downto 0);
    signal muxaalui:std_logic_vector(31 downto 0);
    signal rai:std_logic_vector(31 downto 0);
    signal aluaalu:std_logic_vector(31 downto 0);
    signal cablezero:std_logic;
    signal addamux:std_logic_vector(31 downto 0);
    signal salidapcaux:std_logic_vector(31 downto 0);
    signal retorno:std_logic_vector (31 downto 0);
    signal compuertaand:std_logic;

  begin
    primermux : mux5bits
      port map(
        entrada1 => instruction(20 downto 16),
        entrada2 => instruction(15 downto 11),
        sel => ctrlword(4),
        salida => muxaregistro
      );
    archivoderegistros: Register_File
    port map(
        RegWrite=>ctrlword(1),
        RR1=> instruction (25 downto 21),
        Write_Data: in std_logic_vector (31 downto 0); --En el anterior numero guardar los 8 bytes de esta entrada
        RD1: out std_logic_vector (31 downto 0); --Salida de 32 bits.
        RD2: out std_logic_vector (31 downto 0)); --Salida de 32 bits.


    end component Register_File;

    component alu32bits
    port(
        entrada1:in std_logic_vector(31 downto 0);
        entrada2:in std_logic_vector(31 downto 0);
        control:in std_logic_vector (3 downto 0);
        result: out std_logic_vector(31 downto 0);
        zero:out std_logic
    );
    end component alu32bits;

    component mux5bits
    port(
        entrada1: in std_logic_vector(4 downto 0);
        entrada2: in std_logic_vector(4 downto 0);
        sel: in std_logic;
        salida: out std_logic_vector(4 downto 0)
    );
    end component mux5bits;

    component mux32bits
    port(
        entrada1: in std_logic_vector(31 downto 0);
        entrada2: in std_logic_vector(31 downto 0);
        sel: in std_logic;
```

```
        RR2=> instruction (20 downto 16),
        Write_Register=>muxaregistro,
        Write_Data=>cablememreg,
        RD1=>salida1r,
        RD2=>salida2r,
        clk=>clock,
        reset=>reset
    );

    extensordesigno:sign_ext
    port map(
        entrada16=>instruction(15 downto 0),
        salida32=>se_a_sll2
    );

    shiftleft:sll2
    port map(
        entrada=>se_a_sll2,
        salida=>sll2_a_ADDALU
    );

    muxdespuesregistro:mux32bits
    port map(
        entrada1=>salida2r,
        entrada2=>se_a_sll2,
        sel=>ctrlword(3),
        salida=>muxaalui
    );

    aluimportante: alu32bits
    port map(
        entrada1=>salida1r,
        entrada2=>muxaalui,
        control=>alu_code,
        result=>rai,
        zero=>cablezero
    );

    muxsalidamemoria:mux32bits
    port map(
        entrada1=>rai,
        entrada2=>data_in,
        sel=>ctrlword(2),
        salida=>cablememreg
```

```
      salida: out std_logic_vector(31 downto 0)
   );
   end component mux32bits;

   component sll2
   port(
      entrada: in std_logic_vector(31 downto 0);
      salida: out std_logic_vector(31 downto 0)
   );
   end component sll2;

   component sign_ext
   port(
      entrada16: in std_logic_vector(15 downto 0);
      salida32: out std_logic_vector(31 downto 0)
   );
   end component sign_ext;
   component pc_aux
   port(
      data_in: in std_logic_vector(31 downto 0);
      e: in std_logic;
      clk: in std_logic;
      data_out: out std_logic_vector(31 downto 0);
      reset: in std_logic
   );
   end component pc_aux;

   signal vacio: std_logic;
   signal muxaregistro:std_logic_vector(4 downto 0);
   signal cablememreg:std_logic_vector(31 downto 0);
   signal salida1r:std_logic_vector(31 downto 0);
```

```
   );

   sumador4:alu32bits port map(
      entrada1=>x"00000004",
      entrada2=>salidapcaux,
      control=>"0010",
      result=>aluaalu,
      zero=>vacio
   );

   sumadorsig:alu32bits port map(
      entrada1=>aluaalu,
      entrada2=>sll2_a_ADDALU,
      control=>"0010",
      result=>addamux,
      zero=>vacio
   );

   muxpc: mux32bits
   port map(
      entrada1=>aluaalu,
      entrada2=>addamux,
      sel=>compuertaand,
      salida=>retorno
   );

   pc_pc : pc_aux
      port map(
         data_in => retorno,
         e => '1',
         clk => clock,
         data_out => salidapcaux,
         reset => reset
      );


   instr_addr <= salidapcaux;
   data_out <= salida2r;
   mem_addr <= rai;
   compuertaand<=ctrlword(0) and cablezero;

  end structural ; -- structural
```

re_hab.vhd

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

entity re_hab is
   port(
      data_in: in std_logic_vector(31 downto 0);
      e: in std_logic;
      clk: in std_logic;
      data_out: out std_logic_vector(31 downto 0);
      reset: in std_logic
   );
end re_hab;
architecture behavioral of re_hab is
   begin
   process(clk)
   begin
      if clk'event and clk='1' then
         if (reset = '1' )then
           data_out <= (others => '0');
         elsif e='1' then
            data_out <= data_in;
         end if;
      end if;
   end process;

end behavioral;
```

re_hab_buff.vhd

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

entity re_hab_buff is
   port (
      data_in: in std_logic_vector(31 downto 0);
      eBusA: in std_logic;
      eBusB: in std_logic;
      e: in std_logic;
      clk: in std_logic;
      dataBusA_out: out std_logic_vector(31 downto 0);
      dataBusB_out: out std_logic_vector(31 downto 0);
      reset: in std_logic
   ) ;
  end re_hab_buff;
      data_out: out std_logic_vector(31 downto 0);
      reset: in std_logic
   );
   end component re_hab;


   signal re_buff: std_logic_vector(31 downto 0);
   signal busA: std_logic_vector(31 downto 0);
   signal busB: std_logic_vector(31 downto 0);
  begin

   regi : re_hab
   port map(
     data_in => data_in,
     reset => reset,
 architecture structural of re_hab_buff is

  component tri_state_buff --Buffer de tres estados

    port (
      data_in: in std_logic_vector(31 downto 0);
      data_out: out std_logic_vector(31 downto 0);
      en_buff: in std_logic
    ) ;
  end component tri_state_buff;
  component re_hab

  port(
      data_in: in std_logic_vector(31 downto 0);
      e: in std_logic;
      clk: in std_logic;
      clk => clk,
      e => e,
      data_out => re_buff
    );
    triSateBusA : tri_state_buff
      port map(
        data_in => re_buff,
        data_out => busA,
        en_buff => eBusA
      );
    triSateBusB : tri_state_buff
      port map(
        data_in => re_buff,
        data_out => busB,
        en_buff => eBusB
      );

    dataBusA_out <= busA;
    dataBusB_out <= busB;
  end structural ; -- structural
```

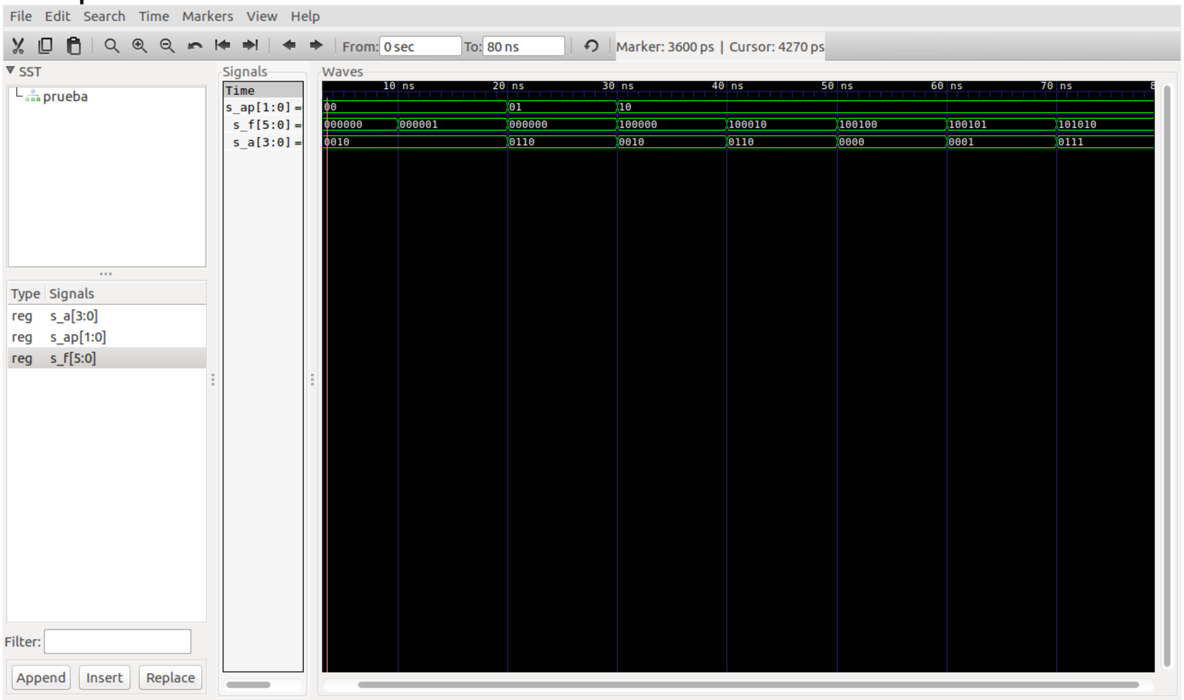
tri_state_buff.vhd

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

entity tri_state_buff is
  port(
      data_in: in std_logic_vector(31 downto 0);
      data_out: out std_logic_vector(31 downto 0);
      en_buff: in std_logic
    );
end tri_state_buff;

architecture behavioral of tri_state_buff is
  begin
    gen_buffer_array : for i in 0 to 31 generate
        data_out(i) <= data_in(i) when (en_buff = '1') else 'Z';
  end generate;
end behavioral;
```

RESULTADOS

Control_alutb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity control_alutb is
end control_alutb;
architecture behavioral of control_alutb
is
    component control_alu
    port(
        ALUOp: in std_logic_vector(1
downto 0);
        FuncCode: in std_logic_vector(5
downto 0);
        ALUCtl: out std_logic_vector(3
downto 0)
    );
    end component;
    signal s_ap: std_logic_vector(1
downto 0);
    signal s_f: std_logic_vector(5
downto 0);
   signal s_a: std_logic_vector(3
downto 0);
    begin
         prueba: control_alu
        port map(
            ALUOp=>s_ap,
 FuncCode=>s_f,
            ALUCtl=>s_a);
   process
        begin
            s_ap <= "00";
            s_f<="000000";
            wait for 10 ns;
            s_f<="000001" ;
            wait for 10 ns;
            s_ap <= "01";
            s_f<="000000";
            wait for 10 ns;
            s_ap <= "10";
            s_f<="100000";
            wait for 10 ns;
            s_f<="100010";
            wait for 10 ns;
            s_f<="100100";
            wait for 10 ns;
            s_f<="100101";
            wait for 10 ns;
            s_f<="101010";
            wait for 10 ns;
            wait;
    end process;
end behavioral;
```



From: 0 sec To: 80 ns

Marker: 3600 ps | Cursor: 4270 ps

Type	Signals
reg	s_a[3:0]
reg	s_ap[1:0]
reg	s_f[5:0]

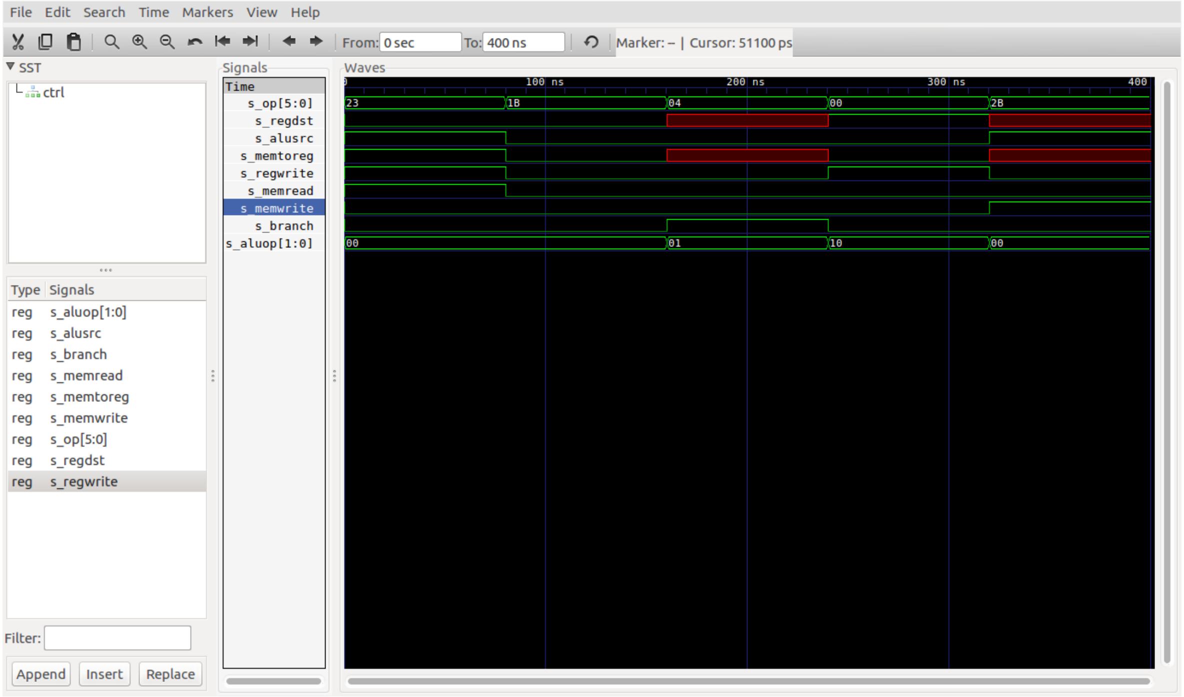
Filter:

Control_unit_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity control_unit_tb is
end control_unit_tb;

architecture behavioral of
control_unit_tb is
    component control_unit
    port(

        op : in std_logic_vector(5 downto
0);
        RegDst, ALUSrc, MemToReg,
RegWrite : out std_logic;
        MemRead, MemWrite, Branch :
out std_logic;
        ALUOp : out std_logic_vector(1
downto 0)
    );
    end component;
```

```
    begin
        ctrl: control_unit
        port map(
            op => s_op,
            RegDst => s_RegDst,
            ALUSrc => s_ALUSrc,
            MemToReg => s_MemToReg,
            RegWrite => s_RegWrite,
            MemRead => s_MemRead,
            MemWrite => s_MemWrite,
            Branch => s_Branch,
            aluop => s_aluop
        );
    process
        begin
            s_op <= "100011"; wait for 80
ns;


            s_op <= "011011"; wait for 80
ns;
```

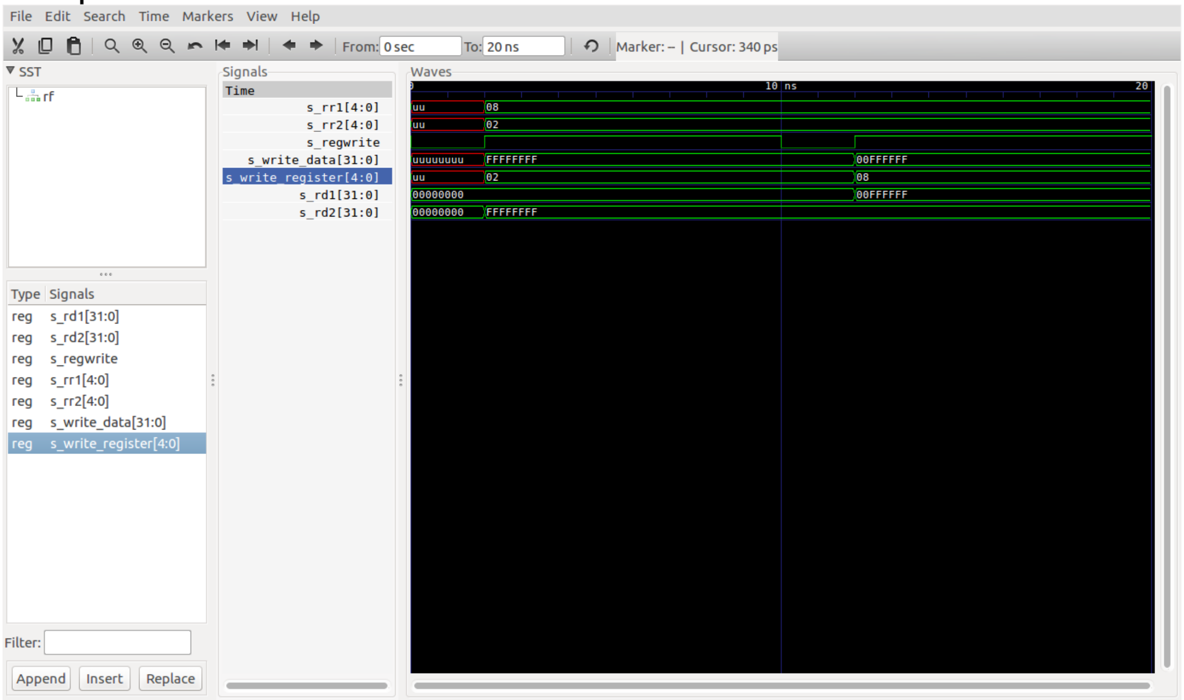
signal s_op: std_logic_vector(5 downto 0);
signal s_RegDst: std_logic;
signal s_ALUSrc: std_logic;
signal s_MemToReg: std_logic;
signal s_RegWrite: std_logic;
signal s_MemRead: std_logic;
signal s_MemWrite: std_logic;
signal s_Branch: std_logic;
signal s_aluop: std_logic_vector(1 downto 0);
	s_op <= "000100"; wait for 80 ns;
	
	s_op <= "000000"; wait for 80 ns;
	
	s_op <= "101011"; wait for 80 ns;
	wait;
	end process;
	end behavioral;



Type	Signals
reg	s_aluop[1:0]
reg	s_alusrc
reg	s_branch
reg	s_memread
reg	s_memtoreg
reg	s_memwrite
reg	s_op[5:0]
reg	s_regdst
reg	s_regwrite

tbRegister_File.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity tbRegister_File is
end tbRegister_File;

architecture behavioral of
tbRegister_File is
   component Register_File
   port(
      RegWrite:in std_logic; --señal de
control
   RR1: in std_logic_vector (4 downto
0); --numero de registro 1
   RR2: in std_logic_vector (4 downto
0); --numero de registro 2
   Write_Register: in std_logic_vector
(4 downto 0);
   Write_Data: in std_logic_vector (31
downto 0); --En el anterior numero
guardar los 8 bytes de esta entrada
   RD1: out std_logic_vector (31
downto 0); --Salida de 32 bits.
   RD2: out std_logic_vector (31
downto 0)
   );
   end component;
   signal s_Write_Data:
std_logic_vector(31 downto 0);
   signal s_RD1: std_logic_vector(31
downto 0);
   signal s_RD2: std_logic_vector(31
downto 0);

begin
rf: Register_File
      port map(
         RegWrite=>s_RegWrite,
         RR1=>s_RR1,
         RR2=>s_RR2,
Write_Register=>s_Write_register,
         Write_Data=>s_Write_Data,
         RD1=>s_RD1,
         RD2=>s_RD2
      );
   process
      begin
         s_RegWrite <= '0';
         wait for 2 ns;
         s_RegWrite <= '1';
         s_RR1<="01000";
         s_RR2<="00010";
         s_Write_Register<="00010";

s_Write_Data<="111111111111111111
11111111111111";
         wait for 8 ns;
         s_RegWrite<='0';
         wait for 2 ns;
         s_RegWrite <= '1';
         s_RR1<="01000";
         s_RR2<="00010";
         s_Write_Register<="01000";

s_Write_Data<="000000001111111111
11111111111111";
         wait for 8 ns;

         wait;
   end process;
end behavioral;
```



Type	Signals
reg	s_rd1[31:0]
reg	s_rd2[31:0]
reg	s_regwrite
reg	s_rr1[4:0]
reg	s_rr2[4:0]
reg	s_write_data[31:0]
reg	s_write_register[4:0]

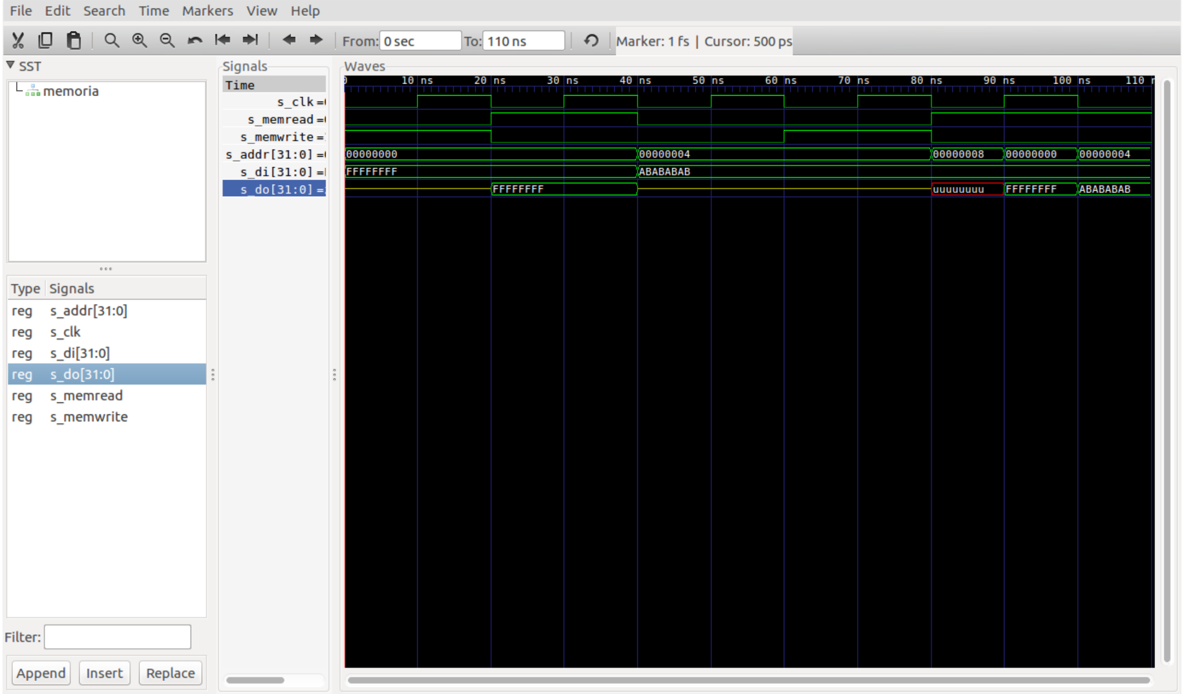
```
Time
s_rr1[4:0]            uu        08
s_rr2[4:0]            uu        02
s_regwrite
s_write_data[31:0]    uuuuuuuu  FFFFFFFF                00FFFFFF
s_write_register[4:0] uu        02                      08
s_rd1[31:0]           00000000                          00FFFFFF
s_rd2[31:0]           00000000  FFFFFFFF
```

Data_memorytb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity data_memorytb is
end data_memorytb;

architecture behavioral of data_memorytb is
    component data_memory
    port(
        clk, MemRead, MemWrite : in std_logic;
    address : in  std_logic_vector(31 downto 0);
    datain  : in  std_logic_vector(31 downto 0);
    dataout : out std_logic_vector(31 downto 0)
    );
    end component;
```

```
s_clk<='0';
s_addr<=x"00000000";
s_MemRead<='0';
s_di<=x"FFFFFFFF";
s_MemWrite<='1';
wait for 10 ns;
s_clk<='1';
wait for 10 ns;
s_clk<='0';
s_MemRead<='1';
s_MemWrite<='0';
wait for 10 ns;
s_clk<='1';
wait for 10 ns;
s_MemRead<='0';
s_clk<='0';
s_MemWrite<='0';
s_addr<=x"00000004";
s_di<=x"ABABABAB";
wait for 10 ns;
s_clk<='1';
```

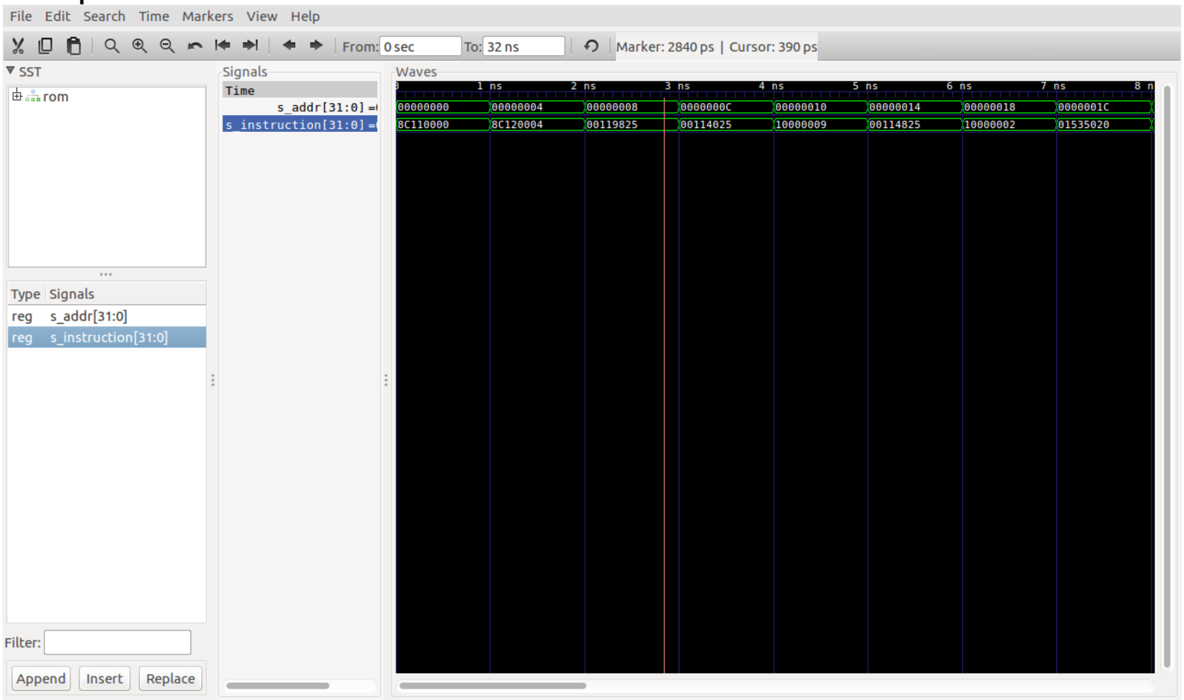
```
    signal s_addr: std_logic_vector(31 downto 0);
    signal s_clk: std_logic;
    signal s_MemRead: std_logic;
    signal s_MemWrite: std_logic;
    signal s_di: std_logic_vector(31 downto 0);
    signal s_do: std_logic_vector(31 downto 0);

    begin
        memoria: data_memory
        port map(
            address => s_addr,
            clk => s_clk,
            MemRead => s_MemRead,
            MemWrite => s_MemWrite,
            datain => s_di,
            dataout => s_do
         );
    process
        begin
            wait for 10 ns;
            s_clk<='0';
            s_MemWrite<='1';
            wait for 10 ns;
            s_clk<='1';
            wait for 10 ns;
            s_clk<='0';
            s_MemWrite<='0';
            s_MemRead<='1';
            s_addr<=x"00000008";
            wait for 10 ns;
            s_clk<='1';
            s_MemRead<='1';
            s_addr<=x"00000000";
            wait for 10 ns;
            s_clk<='0';
            s_MemRead<='1';
            s_addr<=x"00000004";
            wait for 10 ns;

            wait;
    end process;
end behavioral;
```



Type	Signals
reg	s_addr[31:0]
reg	s_clk
reg	s_di[31:0]
reg	s_do[31:0]
reg	s_memread
reg	s_memwrite

Instruction_memorytb.vhd

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

entity instruction_memorytb is
end instruction_memorytb;

architecture mixed of instruction_memorytb is

  component instruction_memory
    port (
      addr: in std_logic_vector(31 downto 0);
      instruction: out std_logic_vector(31 downto 0)
    ) ;
  end component instruction_memory;

  signal s_addr: std_logic_vector(31 downto 0);
signal s_instruction: std_logic_vector(31 downto 0);

begin

  rom: instruction_memory port map(
    addr => s_addr,
    instruction => s_instruction
  );

  stimuli : process
  begin
    for i in 0 to 31 loop
      s_addr <= std_logic_vector(to_unsigned(4*i, s_addr'length));
      wait for 1 ns;
    end loop;
    wait;
  end process stimuli; -- stimuli

end mixed ; -- mixed
```



From: 0 sec To: 32 ns Marker: 2840 ps | Cursor: 390 ps

reg s_addr[31:0]
reg s_instruction[31:0]

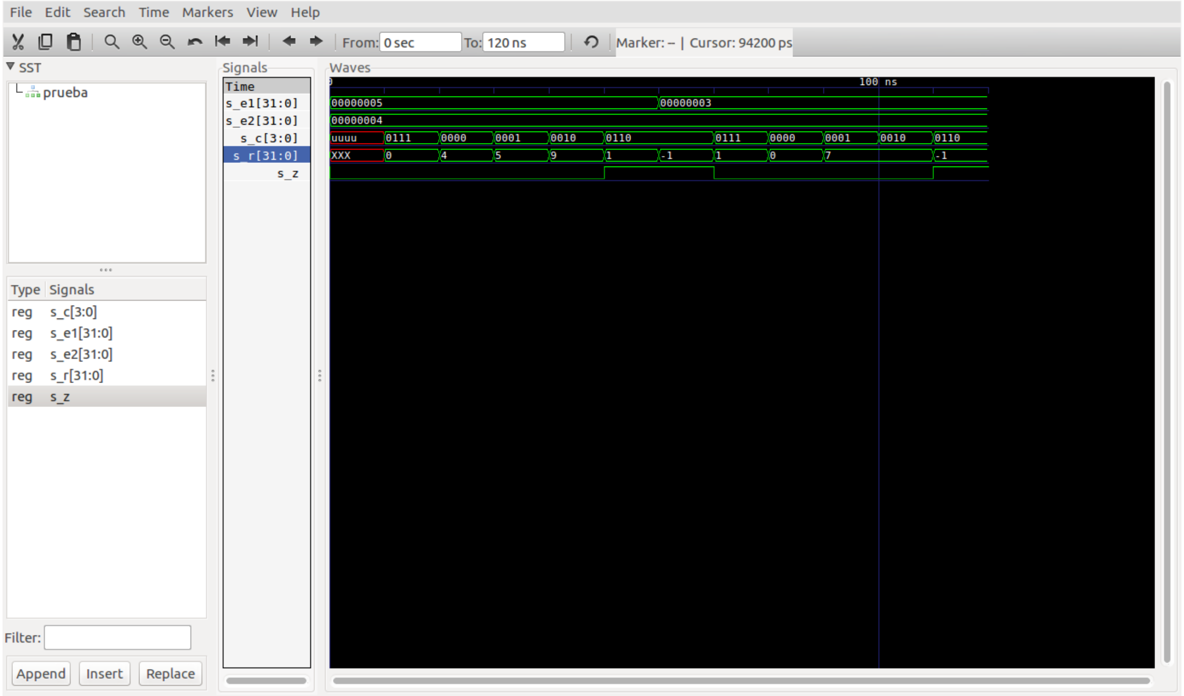
```
Time                 0         1 ns      2 ns      3 ns      4 ns      5 ns      6 ns      7 ns      8 n
s_addr[31:0] =       00000000  00000004  00000008  0000000C  00000010  00000014  00000018  0000001C
s_instruction[31:0] = 8C110000 8C120004  00119825  00114025  10000009  00114825  10000002  01535020
```

Tbalu32bits.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity tbalu32bits is
end tbalu32bits;

architecture behavioral of tbalu32bits is
   component alu32bits
   port(

   entrada1:in std_logic_vector(31 downto 0);
   entrada2:in std_logic_vector(31 downto 0);
   control:in std_logic_vector (3 downto 0);
   result: out std_logic_vector(31 downto 0);
   zero:out std_logic
   );
   end component;
```

```
          entrada2=>s_e2,
          control=>s_c,
          result=>s_r,
          zero=>s_z);

    process
       begin
          s_e1 <= x"00000005";
          s_e2 <= x"00000004";
          wait for 10 ns;
          s_c<="0111";
          wait for 10 ns;
          s_c<="0000";
          wait for 10 ns;
          s_c<="0001";
          wait for 10 ns;
          s_c<="0010";
          wait for 10 ns;
          s_c<="0110";
          wait for 10 ns;
    signal s_e1: std_logic_vector(31
downto 0);
    signal s_e2: std_logic_vector(31
downto 0);
    signal s_c: std_logic_vector(3
downto 0);
    signal s_r: std_logic_vector(31
downto 0);
    signal s_z: std_logic;

    begin
        prueba: alu32bits
        port map(
            entrada1=>s_e1,
```

```
            s_e1 <= x"00000003";
            s_e2 <= x"00000004";
            wait for 10 ns;
            s_c<="0111";
            wait for 10 ns;
            s_c<="0000";
            wait for 10 ns;
            s_c<="0001";
            wait for 10 ns;
            s_c<="0010";
            wait for 10 ns;
            s_c<="0110";
            wait for 10 ns;
            wait;
        end process;
end behavioral;  mixed
```



Type	Signals
reg	s_c[3:0]
reg	s_e1[31:0]
reg	s_e2[31:0]
reg	s_r[31:0]
reg	s_z

SIMULACIÓN FINAL
Processor_testbench.vhd

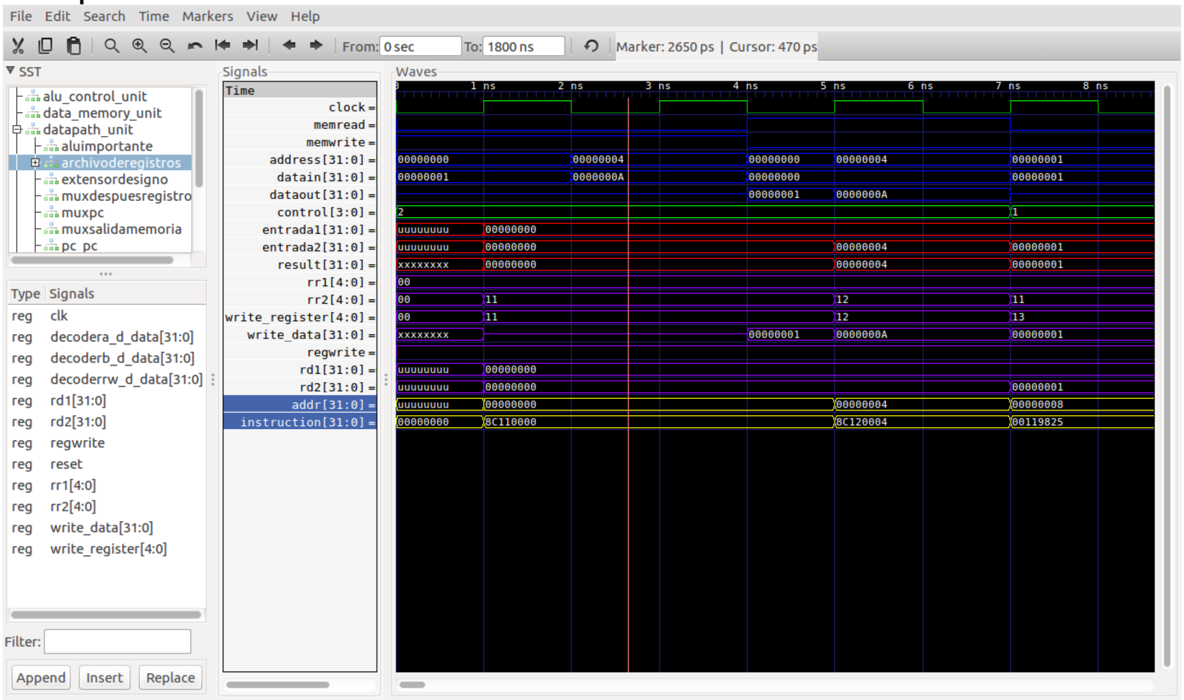
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity processor_testbench is
end processor_testbench;

architecture structural of processor_testbench is

  signal MemRead, MemWrite: std_logic;
  signal initial_data, data, initial_addr, address: std_logic_vector(31 downto 0);
  signal clock: std_logic := '0';
  signal reset: std_logic := '1';
  signal finish_flag: boolean := false;
  signal initial_complete: boolean := false;

  signal instr_addr, instruction_addr, instruction: std_logic_vector(31 downto 0);
  signal mem_data_out, mem_data_in, mem_data_addr: std_logic_vector(31 downto 0);
  --RegDst, ALUSrc, MemToReg, RegWrite, MemRead, MemWrite, PCSrc, ALUOp1 y ALUOp0
  signal ctrlword: std_logic_vector(8 downto 0);
  signal alu_opcode: std_logic_vector(3 downto 0);

  component datapath is
    port (
      data_in: in std_logic_vector(31 downto 0);
      instruction: in std_logic_vector(25 downto 0);
```

```
ctrlword(3);
  MemRead  <= '0' when initial_complete = false else ctrlword(4);
  data <= initial_data when initial_complete = false else mem_data_in;
  address <= initial_addr when initial_complete = false else mem_data_addr;


  datapath_unit: datapath port map(
    data_in => mem_data_out,
    instruction => instruction (25 downto 0),
    ctrlword(4) => ctrlword(8), --RegDest
    ctrlword(3) => ctrlword(7), --ALUSrc
    ctrlword(2) => ctrlword(6), --MemToReg
    ctrlword(1) => ctrlword(5), --RegWrite
    ctrlword(0) => ctrlword(2), --Branch
    alu_code => alu_opcode,
    data_out => mem_data_in,
    mem_addr => mem_data_addr,
    instr_addr => instruction_addr,
    clock => clock,
    reset => reset
  );

  alu_control_unit: control_alu port map(
    ALUOp => ctrlword(1 downto 0),
    FuncCode => instruction(5 downto 0),
    ALUCtl => alu_opcode
  );

  unidadcontrol: control_unit port map(
    op => instruction(31 downto 26),
    RegDst => ctrlword(8),
      --RegDst, ALUSrc, MemToReg,
RegWrite, branch
      ctrlword: in std_logic_vector(4
downto 0);
      alu_code: in std_logic_vector(3
downto 0);
      data_out, mem_addr: out
std_logic_vector(31 downto 0);
      instr_addr: out std_logic_vector(31
downto 0);
      clock, reset: in std_logic
    );
  end component datapath;

  component instruction_memory
    port (
      addr: in std_logic_vector(31
downto 0);
      instruction: out std_logic_vector(31
downto 0)
    ) ;
  end component instruction_memory;

  component data_memory
    port(
      clk, MemRead, MemWrite : in
std_logic;
      address : in  std_logic_vector(31
downto 0);
      datain  : in  std_logic_vector(31
downto 0);
      dataout : out std_logic_vector(31
downto 0)
  );
  end component data_memory;

  component control_unit
  port (
    op : in std_logic_vector(5 downto 0);
    RegDst, ALUSrc, MemToReg,
RegWrite : out std_logic;
    MemRead, MemWrite, Branch : out
std_logic;
```

```
    ALUSrc => ctrlword(7),
    MemToReg => ctrlword(6),
    RegWrite => ctrlword(5),
    MemRead => ctrlword(4),
    MemWrite => ctrlword(3),
    Branch => ctrlword(2),
    ALUOp => ctrlword(1 downto 0)
  );

  instruction_memory_unit:
instruction_memory port map(
    addr => instruction_addr,
    instruction => instruction
  );

  data_memory_unit: data_memory port
map(
    clk => clock,
    MemRead => MemRead,
    MemWrite => MemWrite,
    address => address,
    datain => data,
    dataout => mem_data_out
  );



  clk_gen : process
  begin
    wait for 1 ns;
    clock <= not clock;
    if finish_flag = true then
      wait;
    end if;
  end process clk_gen; -- clk_gen

  reset_p : process
  begin
    wait for 4 ns;
    reset <= '0';
    wait;
  end process reset_p; -- reset_p

  write_mem : process
  begin
```

```
    ALUOp : out std_logic_vector(1 downto 0)
  );
  end component control_unit;

  component control_alu
    port (
      ALUOp : in std_logic_vector(1 downto 0);
      FuncCode : in std_logic_vector(5 downto 0);
      ALUCtl : out std_logic_vector(3 downto 0)
    );
  end component control_alu;



begin

  MemWrite <= '1' when initial_complete = false else
```

```
    initial_addr <= x"00000000";
    initial_data <= x"00000001"; -- 1
    wait for 2 ns;
    initial_addr <= x"00000004";
    initial_data <= x"0000000a"; -- 1
    wait for 2 ns;
    initial_complete <= true;
    wait;
  end process ; -- write_mem

  stimuli : process
  begin
    while initial_complete = false loop
      wait for 1 ns;
    end loop ;
    processing : while instruction /= x"00000000" loop
      wait for 1 ns;
    end loop ; -- processing
    finish_flag <= true;
    wait;
  end process stimuli; -- stimuli

end structural ; -- structural
```



Conclusion

Para un programador sin mucha experiencia en lenguajes de descripción de hardware esto fue una travesia. Como se ha visto en clase la arquitectura MIPS en procesadores es ampliamente usada, mas en la década de los noventas. Su implementación en VHDL tiene su truco, como podemos observar muchos modulos están programados de manera “Behavioral” y otros “Structural” y esto tiene su razón de ser. A la hora del desarrollo y al evaluar cada modulo individualmente no tuve ningún problema, sin embargo a la hora de simular todo el procesador me enfrento a que no hay lo que tenia que hacer, las instrucciones no salían adecuadamente, dicho problema lo resolvi pero ahora me enfrente a que mi Register_File no otorgaba los valores de salida, en el desarrollo se puede apreciar que hay dos Register_File, el primero programado de una manera similar a la Data Memory el siguiente Register_File_v2 programado de una manera más estructural. Aquí entra un aprendizaje de este proyecto, el “alto nivel” en VHDL a pesar de ser muy comodo no es lo más efectivo (al menos en este proyecto) ya que al desarrollar Register_File (y la ALU32bits) quice ahorrar tiempo programándolo de manera Funcional y no Estructural, debido a esto es que mis primeras simulaciones no prosperaron.Al investigar como hacer el Register_File me encuentro con modelos pensados en bajo nivel asi nace Register_File_v2 y gracias a la implementación de este es que la simulación corre.

Sin embargo me encuentro con otro contratiempo dentro de la simulación, las direcciones que entran a addres de Data_Memory estaban incorrectas a lo que la ALU otorgaba y eso lo resolvi modificando un cable mal conectado al MUX que recibe la salida del sign_extender. Es asi como la simulación corre exitosamente, las instrucciones, las direcciones, entradas y salidas de diversos modulos corresponden con la instrucción entrante. En la captura de pantalla adjunte las señales de los modulos que considere más importantes en azul se encuentran las señales del data_memory, en rojo las señales de la ALU32bits, en morado las del Register_File y en amarillo las señales del PC_Aux. Es importante recalcar que la ALU32bits la deje en una programación “Comportamiento” ya que hace su labor correctamente sin embargo creo que seria mejor una programación del tipo Estructural como la del Register_File. En conclusión, al programar en VHDL un dispositivo como un procesador con cierta arquitectura es siempre mejor usar el modo Estructural para algunos modulos, el ganar tiempo programando el comportamiento de los modulos no es lo ideal aquí. Para finalizar aparte de la simulacion final adjunte también la simulacion de cada uno de los modulos individualmente, en el caso del Register_File deje la primera simulacion ya que el funcionamiento del Register_File_v2 se veria en la simulacion del procesador completo.