

Práctica de Organización del Computador I

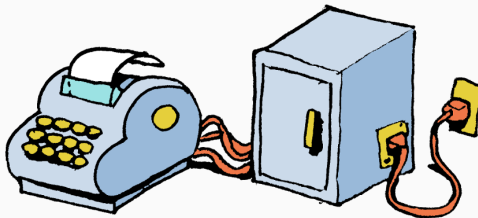
Introducción

Segundo Cuatrimestre 2022

Organización del Computador I
DC - UBA

Motivación

El objetivo de la práctica de Organización del Computador I es **comprender y apropiarse de los principios de diseño e implementación de un soporte electrónico que nos permita implementar un modelo de cómputo sobre el cual ejecutar nuestros programas.**





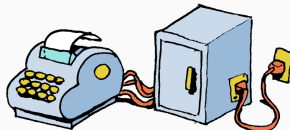
(fenómenos naturales)

↓ **modelado** ↑ **interpretación**

$$Q = \begin{bmatrix} 0,1 & 0 \\ 0 & 0,1 \end{bmatrix}$$

(modelo)

↓ **ejecución** ↑ **verificación**



(cómputo)

¿Qué precisamos para modelar/implementar/ejecutar los fenómenos observables?

Representación de la información , una forma de representar los datos con los que vamos a trabajar (números)

Conservación de los datos , un mecanismo que conserve el valor de nuestros datos, salvo que indiquemos lo contrario, a través del tiempo (memorias)

Modelo de cómputo , una definición de cómo operar con la información representada (set de instrucciones, microoperaciones)

¿Qué precisamos para modelar/implementar/ejecutar los fenómenos observables?

Operaciones aritmético/lógicas , una implementación de operaciones aritméticas y lógicas básicas sobre nuestra información representada (ALU)

Cómputo en base a valores previos , una forma de resolver operaciones complejas a partir de ejecutar operaciones sencillas (atómicas) en una serie de pasos (circuitos secuenciales, microprogramación)

Recordemos que para poder computar precisamos:
Representar , Conservar , Operar



Recordemos que para poder computar precisamos:
Enteros acotados , Memorias , ALU



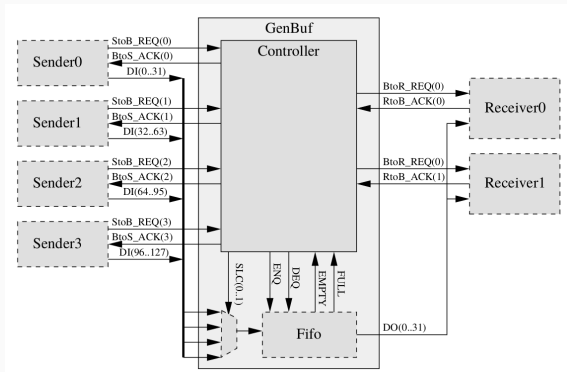
Hardware - su práctica

Al igual que el software, la práctica del diseño y la implementación del hardware depende del siguiente grupo de actividades:

Diseño → Especificación → Implementación → Validación → Verificación

Del diseño a la implementación tenemos:

Diseño Descripción de arquitectura, división por vistas del sistema



Del diseño a la implementación tenemos:

Especificación Representación declarativa del comportamiento esperado

G1	$\forall i : \text{always} (\text{StoB_REQ}(i) \rightarrow \text{eventually! BtoS_ACK}(i))$
	$\forall i : \text{always} (\neg \text{StoB_REQ}(i) \rightarrow \text{eventually! } \neg \text{BtoS_ACK}(i))$
G2	$\forall i : \text{always} (\text{rose}(\text{StoB_REQ}(i)) \rightarrow \neg \text{BtoS_ACK}(i))$
G3	$\forall i : \text{always} (\text{rose}(\text{BtoS_ACK}(i)) \rightarrow \text{prev}(\text{StoB_REQ}(i)))$
G4	$\forall i : \text{always} ((\text{BtoS_ACK}(i) \wedge \text{StoB_REQ}(i)) \rightarrow \text{next! BtoS_ACK}(i))$
A1	$\forall i : \text{always} (\text{StoB_REQ}(i) \wedge \neg \text{BtoS_ACK}(i) \rightarrow \text{next! StoB_REQ}(i))$
	$\forall i : \text{always} (\text{BtoS_ACK}(i) \rightarrow \text{next! } \neg \text{StoB_REQ}(i))$
G5	$\forall i \forall i' \neq i : \text{always} \neg (\text{BtoS_ACK}(i) \wedge \text{BtoS_ACK}(i'))$
A2	$\forall j : \text{always} (\text{BtoR_REQ}(j) \rightarrow \text{eventually! RtoB_ACK}(j))$
	$\forall j : \text{always} (\neg \text{BtoR_REQ}(j) \rightarrow \text{next! } \neg \text{RtoB_ACK}(j))$
A3	$\forall j : \text{always} (\text{BtoR_REQ}(j) \wedge \text{RtoB_ACK}(j) \rightarrow \text{next! RtoB_ACK}(j))$
A4	$\forall j : \text{always} (\text{RtoB_ACK}(j) \rightarrow \text{prev}(\text{BtoR_REQ}(j)))$

Del diseño a la implementación tenemos:

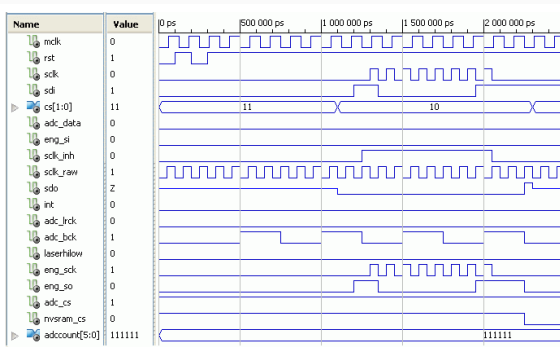
Implementación

Interpretación imperativa
de la especificación que
permite producir una
instancia operacional del
sistema

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity signed_adder is
6   port
7   (
8     aclr : in    std_logic;
9     clk  : in    std_logic;
10    a     : in    std_logic_vector;
11    b     : in    std_logic_vector;
12    q     : out   std_logic_vector
13  );
14 end signed_adder;
15
16 architecture signed_adder_arch of signed_adder is
17   signal q_s : signed(a'high+1 downto 0); -- extra bit wide
18
19 begin -- architecture
20   assert(a'length >= b'length)
21     report "Port A must be the longer vector if different sizes!"
22     severity FAILURE;
23   q <= std_logic_vector(q_s);
24
25   adding_proc:
26   process (aclr, clk)
27   begin
28     if (aclr = '1') then
29       q_s <= (others => '0');
30     elsif rising_edge(clk) then
31       q_s <= ('0' & signed(a)) + ('0' & signed(b));
32     end if; -- clk'd
33   end process;
34
35 end signed_adder_arch;
```

Y en las etapas post implementativas:

Validación Conjunto de pruebas no exhaustivas que prueban el comportamiento de la especificación y/o implementación



Y en las etapas post implementativas:

Verificación Prueba de propiedades formales (con garantías basadas en algún mecanismo matemático) de la especificación y/o implementación

```
import "DwyerPatterns.spectra"

module ElevatorUnrealizable

type Floors = Int(0..3);
sys Floors elevatorLocation;
env Floors request;

gar startOnGroundFloor:
  elevatorLocation=0;

gar moveOneFloorAtATime:
  G (next(elevatorLocation) = elevatorLocation+1) |
    (next(elevatorLocation) = elevatorLocation-1);

gar eventuallyHandleOpenRequests:
  pRespondsToS(request != elevatorLocation,
               request=elevatorLocation);
```

En la práctica de la materia vamos a concentrarnos en las etapas de **diseño** y **implementación**, la primera al estudiar las arquitecturas existentes, sus motivaciones y prácticas comunes, y la segunda al construir una instancia de un procesador de 16 bits sobre una arquitectura de Von Neumann. Aunque no diseñemos nuestros propios tests, la ejecución de benchmarks sobre los componente escritos son una forma de **validación**.

Conceptos principales

A lo largo de la práctica vamos a emplear y re-interpretar los siguientes conceptos principales:

Razonamiento composicional describe la práctica de dividir el funcionamiento del sistema en componentes que se encargan de una parte menor del comportamiento

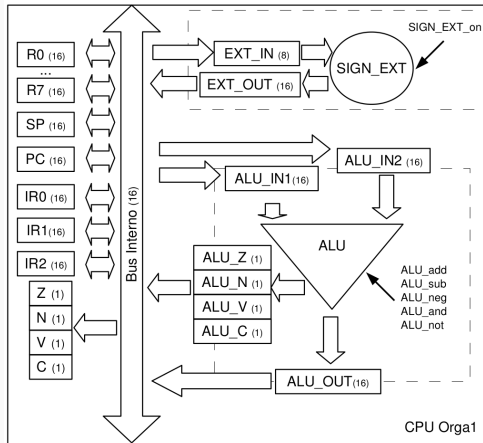
Abstracción es el proceso a través del cual ocultamos detalles de un sistema o componente de acuerdo a las necesidades de la tarea u operación en curso

Diseño por contratos describe la forma en la que vinculamos componentes a partir de derechos de los que el componente goza y obligaciones que debe cumplir

Recordemos que queremos diseñar e implementar un modelo de cómputo (microprocesador). Para el alcance de la materia vamos a interpretar a la arquitectura del mismo como:

Diseño de sistema + Set de instrucciones → Microarquitectura

Diseño de sistema Elección de arquitectura y componentes involucrados



Set de instrucciones Semántica de las capacidades de ejecución de nuestro microprocesador

Formato de instrucción

Tipo 1: Instrucciones de dos operandos

4 bits	6 bits	6 bits	16 bits	16 bits
cod. op.	destino	fuelle	constante destino (opcional)	constante fuente (opcional)

operación	cod. op.	efecto
MOV d, f	0001	$d \leftarrow f$
ADD d, f	0010	$d \leftarrow d + f$ (suma binaria)
SUB d, f	0011	$d \leftarrow d - f$ (resta binaria)
AND d, f	0100	$d \leftarrow d \text{ and } f$
OR d, f	0101	$d \leftarrow d \text{ or } f$
CMP d, f	0110	Modifica los <i>flags</i> según el resultado de $d - f$ (resta binaria)
ADDC d, f	1101	$d \leftarrow d + f + \text{carry}$ (suma binaria)

Microarquitectura Implementación de nuestro set de instrucciones a través de la lógica de control de la arquitectura y componentes seleccionados

► MOV R5,R1

1. R5 := R1

► AND R7, R1

1. ALU_IN1 := R7
2. ALU_IN2 := R1
3. ALU_{and}
4. R7 := ALU_OUT
5. Z := ALU_Z
6. N := ALU_N
7. C := ALU_C
8. V := ALU_V

► JE 0xFF

1. IF Z = 1
2. EXT_IN := IR0[7:0]
3. SIGN_EXT_on
4. ALU_IN_1 := PC
5. ALU_IN_2 := EXT_OUT
6. ALU_{add}
7. PC := ALU_OUT

Contenidos de la práctica

Los contenidos que vamos a presentar durante las clases prácticas son los siguientes:

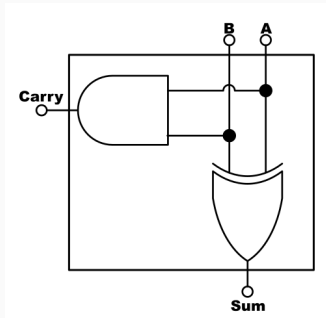
Lógica Combinatoria y Secuencial

Diseño de un set de instrucciones

Microprogramación

Manejo de memoria, Interrupciones, Caché y Buses

Lógica Combinatoria presenta los principios fundamentales para construir circuitos que implementen en un soporte electrónico la semántica de la lógica proposicional



El diseño de un set de instrucciones es el paso necesario para implementar nuestro microprocesador, donde vamos a definir el nombre y significado de las operaciones que deseamos poder ejecutar como parte de nuestros programas

La microprogramación va a describir la forma en que nuestros componentes sincrónicos interactúan para implementar las operaciones descritas en el set de instrucciones previamente definido

Estructura de la práctica

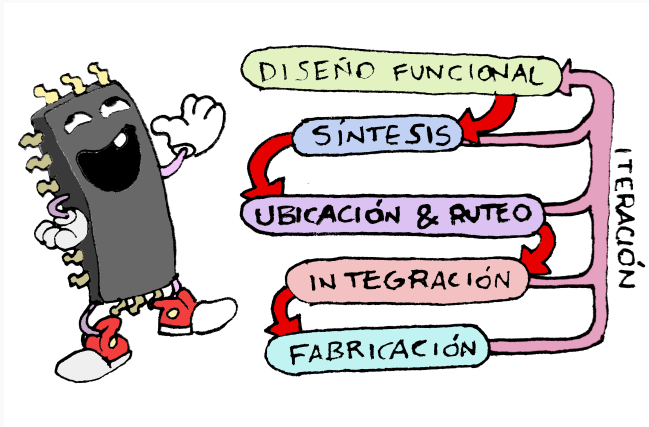
Las clases prácticas van a dividirse de la siguiente manera:

Clases expositivas donde vamos a repasar temas presentados en la teórica

Talleres a partir de los cuales vamos a ir definiendo y probando, en un lenguaje de especificación de hardware (HDL), los componentes necesarios para construir nuestro microprocesador

Un evaluación escrita donde van a tener que diseñar, analizar y/o implementar algunos elementos de una arquitectura distinta a la que trabajamos en el taller

HDL



Del diseño a la implementación tenemos:

Un lenguaje de descripción de hardware, o HDL por sus siglas en inglés (hardware description language) es un lenguaje que describe la estructura y el comportamiento de un circuito digital. Los dos lenguajes más utilizados en la industria son VHDL y Verilog. En Orga 1 vamos a usar **VHDL**.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity signed_adder is
6   port
7   (
8     aclr : in    std_logic;
9     clk  : in    std_logic;
10    a     : in    std_logic_vector;
11    b     : in    std_logic_vector;
12    q     : out   std_logic_vector
13  );
14 end signed_adder;
15
16 architecture signed_adder_arch of signed_adder is
17   signal q_s : signed(a'high+1 downto 0); -- extra bit wide
18
19 begin -- architecture
20   assert(a'length >= b'length)
21     report "Port A must be the longer vector if different sizes!"
22     severity FAILURE;
23   q <= std_logic_vector(q_s);
24
25   adding_proc:
26   process (aclr, clk)
27   begin
28     if (aclr = '1') then
29       q_s <= (others => '0');
30     elsif rising_edge(clk) then
31       q_s <= ('0'&signed(a)) + ('0'&signed(b));
32     end if; -- clk'd
33   end process;
34
35 end signed_adder_arch;
```


Un documento VHDL va estar dividido en una descripción de interfaz y otra de comportamiento. La descripción de interfaz se define como **entity**, da un nombre al componente y enumera los tipos de señales expuestas, indicando su tamaño y si se trata de una entrada o una salida.

```
entity signed_adder is
port
(
    aclr : in    std_logic;
    clk  : in    std_logic;
    a    : in    std_logic_vector;
    b    : in    std_logic_vector;
    q    : out   std_logic_vector
);
end signed_adder;
```

La descripción de comportamiento se define como **architecture** y define el vínculo funcional y/o lógico entre las señales de entrada, salida y cualquier elemento de representación interna, que se puede definir de forma estática o, por ej., en base al evento de flanco de reloj.

```
architecture signed_adder_arch of signed_adder is
    signal q_s : signed(a'high+1 downto 0); -- extra bit wide

begin -- architecture
    assert(a'length >= b'length)
        report "Port A must be the longer vector if different size"
        severity FAILURE;
    q <= std_logic_vector(q_s);

    adding_proc:
    process (aclr, clk)
    begin
        if (aclr = '1') then
            q_s <= (others => '0');
        elsif rising_edge(clk) then
            q_s <= ('0'&signed(a)) + ('0'&signed(b));
        end if; -- clk'd
    end process;
end signed_adder_arch;
```

Cierre

En esta presentación vimos:

Motivación Necesidad de modelar, implementar y computar

Hardware - su práctica Diseño, especificación, implementación, validación y verificación

Conceptos principales 1 Razonamiento composicional, abstracción, diseño por contratos

Conceptos principales 2 Diseño de sistema, set de instrucciones, microarquitectura

Contenidos de la práctica Lógica combinatoria y secuencial, diseño de un set de instrucciones, microprogramación, memoria, interrupciones, caché y buses

Estructura de la práctica Clases, talleres, evaluación

HDL Fases de fabricación del software, descripción y estructura de VHDL