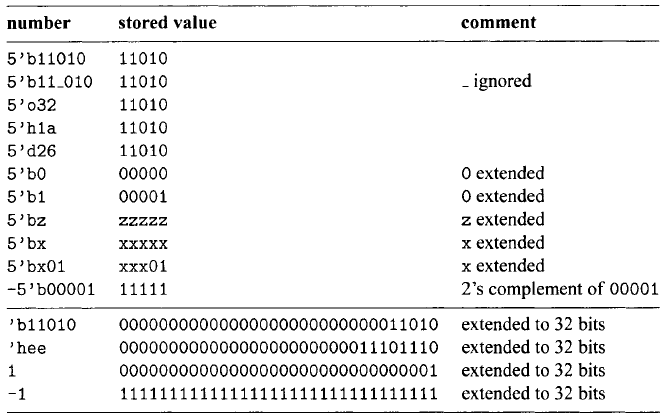
## Representación de números

[sign] [size] ' [base] [value]

Donde:

* Size: Especifica la cantidad de bits del numero. Puede ser omitido si es un número decimal.
* Base:
  + b or B: binary
  + o or O: octal
  + h or H: hexadecimal
  + d or D: decimal
* Value: Valor del número de la base correspondiente

Ejemplos



## Declaración de módulo y puertos

**module** [module-name]

(

[mode] [data-type] [port-names] ,

. . .

[mode] [data-type] [port-names]

);

Donde:

* Module-name. Es el nombre del módulo
* Mode. Puede ser:
  + Input
  + Output
  + Inout
* Data-type. En caso de omisión es wire. Puede ser:
  + Wire
  + Reg
* Port-names. Son los nombres que queremos asignar a los puertos

Ejemplo:

module ALU(input [5:0]codigo,

input [`TAMANIO:0]a,

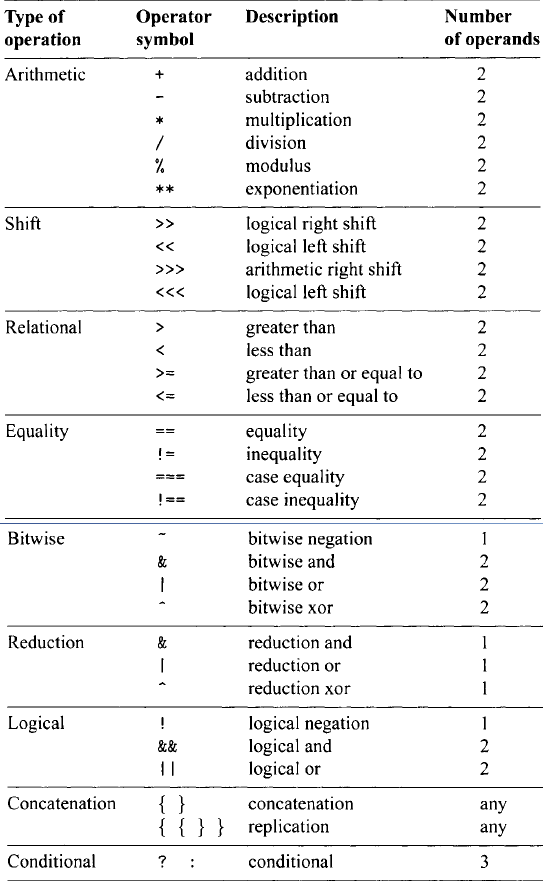
input [`TAMANIO:0]b,

output reg [`TAMANIO:0]c

);

## Operadores

Detalles de los operadores en pag. 70 del PDF Chu



## Parametros y constantes

**Constantes.** Funcionan a nivel global

localparam DATA-WIDTH = 8 ,

DATA-RANGE = 2\*\*DATA\_WIDTH - 1;

localparam UART-PORT = 4'b0001,

LCD-PORT = 4'b0010,

MOUSE-PORT = 4'b0l00;

**Parametros**. Funcionan a nivel de módulo.

module [module-name]

#(

parameter [parameter-name] = [default-value] ;

[parameter-name] = [default-value] ;

)

(

. . . // I/0 port declaration

)

## Declaración de señales

[data-type] [port-names];

Ejemplo:

wire p0, p1;

wire [7:0] datal , data2; //8-bitdata

wire [31:0] addr; // 32-bit address

wire [0:7] revers-data; //ascending index should be avoided

wire [3:0] meal [31:0] ; //32-by-4 memory

Declaración implícita (por defecto, wire):

assign p1 = i0 & i1;

## Cuerpo del programa

Puede tener:

* Asignación continua

assign[signal-name] = [expression] ;

* Bloques always. Contiene sentencias que se ejecutan ordenadas secuencialmente para facilitar su codificación, pero que en tiempo de ejecución se comportan como cajas negras combinacionales.
* Instanciación de modulos.

[module-name] [instance-name]

(

.[port-name] ([signal-name]) ,

.[port-name] ([signal-name]),

...

);

Ejemplo

eq1 eq-bitl-unit (. eq(el), . i0(a[l]), . il(b[11]) ;

(De esta forma, no necesariamente tienen que estar ordenados los puertos.)

## Bloques Always

always @([sensitivity-list])

begin [optional name]

[optional local variable declaration];

[procedural statement];

[procedural statement] ;

End

Donde:

Sensitivity list. Es la lista de señales a las que responde el bloque always. Es decir, cuando cambia alguna de ellas, el bloque se ejecuta. We can use always @\* to implicitly include all the input signals.

**Asignaciones**

A procedural assignment can only be used within an always block or initial block. There are two types of assignments: blocking assignment and nonblocking assignment. Their

basic syntax is:

[variable-name] = [expression] ; // blocking assignment

[variable-name] <= [expression] ; // nonblocking assignment

In a blocking assignment, the expression is evaluated and then assigned to the variable immediately, before execution of the next statement (the assignment thus "blocks" the execution of other statements). It behaves like the normal variable assignment in the C language. In a nonblocking assignment, the evaluated expression is assigned at the end of the always block (the assignment thus does not block the execution of other statements).

**Sentencia IF**

if [boolean-expr]

begin

[proceduralstatement];

[proceduralstatement];

end

else

begin

[proceduralstatement];

[proceduralstatement];

end

**Sentencia CASE**

case [case-expr]

[item] :

begin

[procedural statements] ;

. . .

end

[item] :

begin

[procedural statements] ;

. . .

end

default :

begin

[procedural statements] ;

. . .

end

endcase

## Testbench

`timescale 1ns / 1ps //duración de la simulación / duración de cada tiempo

module test\_TP2;

// Declaro Señales

reg [7:0] registro;

reg s1;

reg s2;

reg s3;

wire [7:0] salida;

// Instantiate the Unit Under Test (UUT)

TP2 uut (

.registro(registro),

.s1(s1),

.s2(s2),

.s3(s3),

.salida(salida)

);

//aquí pruebo los “vectores” que quiero colocar como inpts, separando en tiempos sucesivos mediante el caracter #

initial begin

registro = 0;

s1 = 0;

s2 = 0;

s3 = 0;

#10; //espero 10 unidades de tiempo

(…)//continua

end

endmodule

## Circuitos Secuenciales

**Flip Flop D**

module d\_ff

(

input wire clk,

input wire d ,

output reg q

);

// body

always @(posedge clk)

q <= d;

endmodule

**Flip Flop D con reset asincrono**

module d\_ff

(

input wire clk,reset

input wire d ,

output reg q

);

// body

always @(posedge clk, posedge reset)

if(reset)

q <= 1’b0;

else

q <= d;

endmodule

**Flip Flop D con reset asincrono**

module d\_ff

(

input wire clk,reset

input wire d ,

input wire en ,

output reg q

);

// body

always @(posedge clk, posedge reset)

if(reset)

q <= 1’b0;

else if (en)

q <= d;

endmodule