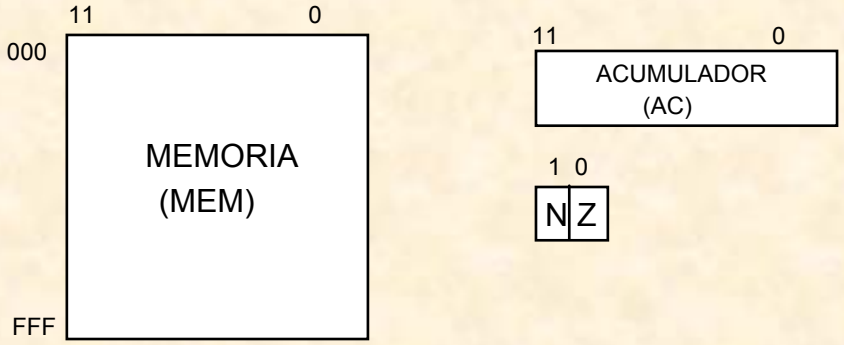
Sesión 8: Diseño de un Computador con VHDL

VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Definición del Computador

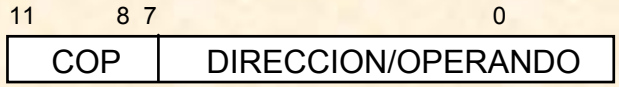
Memoria y registros

- Memoria principal de 256 palabras de 12 bits (256x12)
- Un registro acumulador (AC) también de 12 bits.
- Registro de códigos de condición de 2 bits: Z y N



Formato de las instrucciones

- Todas las instrucciones tienen longitud fija (12 bits)
- Están compuestas por:
  - código de operación (COP) de 4 bits (los 4 más significativos)
  - dirección o un operando de 8 bits (los menos significativos).



VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Repertorio de instrucciones

11 instrucciones:

- 3 de carga y almacenamiento (LDA, STA, LDAI)
- 3 aritmético-lógicas (SUM, SUMI, NOR)
- 4 de salto condicional (JZ, JNZ, JN, JNN)
- 1 instrucción de parada (HALT)

Nombre Simbólico	DIR/OPE	COP	Significado
LDA	DIR	0001	AC <-- MEM(DIR)
STA	DIR	0010	MEM(DIR) <-- <AC>
SUM	DIR	0011	AC <-- <AC> + MEM(DIR)
LDAI	OPE	0100	AC <-- 0000&OPE
SUMI	OPE	0101	AC <-- <AC> + 0000&OPE
NOR	DIR	0110	AC <-- <AC> NOR MEM(DIR)
JZ	DIR	0111	PC <-- DIR SI <Z> = 1
JNZ	DIR	1000	PC <-- DIR SI <Z> = 0
JN	DIR	1001	PC <-- DIR SI <N> = 1
JNN	DIR	1010	PC <-- DIR SI <N> = 0
HALT	-	0000	parada

VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Ejemplos de programas

Ejemplo 1
Suma 3 al contenido de la posición de memoria 4 y almacena el resultado en la posición de memoria 5.

Simbólico	Binario
LDAI 3	0100 00000011
SUM 4	0011 00000100
STA 5	0010 00000101
HALT	0000 00000000

Para referenciar posiciones de memoria consecutivas (indexación) modificamos instrucciones en tiempo de ejecución, concretamente, sumando un 1 a la instrucción *STA índice* (inicialmente en binario 001000010011) de la posición 4.

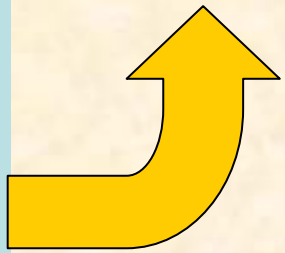
De esa forma, cada vez que se recorre el ciclo que constituye el programa, la instrucción *STA índice* hace referencia a la dirección siguiente a la que hizo en el recorrido anterior.

El ciclo se controla comprobando cuando el *índice* toma el valor *límite*.

Para ello se realiza la resta *limite - índice*, complementando a dos *índice* (complemento a 1 más 1) y sumando el resultado a *índice*

Ejemplo 2
Inicializa 10 posiciones de memoria (de la 20 a la 29) con un contenido igual a su dirección.

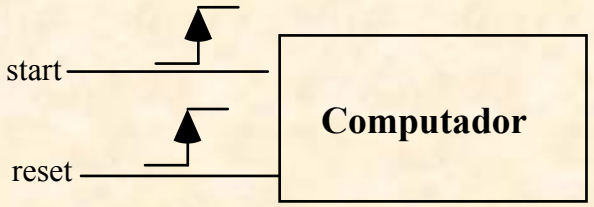
	Simbólico	Binario
0	LDA (STA índice)	0001 00000100
1	SUMI 1	0101 00000001
2	STA (STA índice+1)	0010 00000100
3	LDA índice	0001 00001110
4	STA índice	0010 00010011
5	SUMI 1	0101 00000001
6	STA índice	0010 00001110
7	LDA limite	0001 00001101
8	NOR limite	0110 00001101
9	SUMI 1	0101 00000001
10	SUM índice	0011 00001110
11	JNZ 0	1000 00000000
12	HALT	0000 00000000
13	limite	000000011110
14	índice	000000010100



VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo de Comportamiento (1)

- Utiliza una entidad con un único proceso
- La entidad dispone de dos puertos binarios de entrada:
  - *Start:* para iniciar la ejecución de un programa
  - *Reset:* para reinicializar la máquina (poner a cero el contador de programa).
- Ambas entradas deberán activarse con flancos positivos.




```
ENTITY computador IS
   PORT (start, reset : IN bit);
END computador;
```

Elemetos de modeloado (comportamiento):

- Zonas de datos (registros y memoria): variables
- Transferencias: asignación de variables
- Transformaciones: operadores y funciones
- Control del flujo: sentencias *if* y *case*.

- Para la memoria principal de 256x12 bits se utiliza un bit_vector de longitud 12 sobre el rango de los naturales (0 a 255).
- La carga inicial del programa a ejecutar se define como el valor inicial de la variable que soporta la memoria (mem):

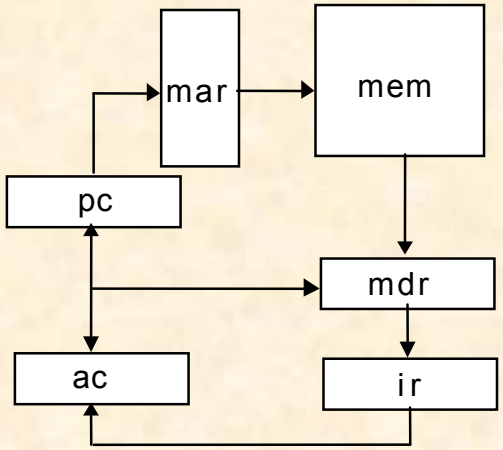
```
ARRAY (NATURAL RANGE 0 TO 255) OF bit_vector(11 DOWNTO 0);
VARIABLE mem : array_memoria :=
-- programa a ejecutar
("000100000100", --  0 LDA (STA indice)
 "010100000001", --  1 SUMI 1
 "001000000100", --  2 STA (STA indice+1)
 "000100001110", --  3 LDA indice
 "001000010011", --  4 STA indice
 "010100000001", --  5 SUMI 1
 "001000001110", --  6 STA indice
 "000100001101", --  7 LDA limite
 "011000001101", --  8 NOR limite
 "010100000001", --  9 SUMI 1
 "001100001110", -- 10 SUM indice
 "100000000000", -- 11 JNZ 0
 "000000000000", -- 12 HALT
 "000000011110", -- 13 limite
 "000000010100", -- 14 indice
  OTHERS => "000000000000");
```

Modelo de Comportamiento (2)

- Además de los registros visibles por el programador, esto es, *ac* y *nz*, utilizaremos variables para los registros internos que tendrán existencia real en la máquina:
  - registro contador de programa (*pc*)
  - registro de instrucciones (*ir*)
  - registro de direcciones de memoria (*mar*)
  - registro de datos de memoria (*mdr*).

- Fases en la ejecución de una instrucción:
  Búsqueda
  - lee de memoria la instrucción apuntada por el *pc*, es decir, se transferirá el contenido del *pc* al *mar*
  - lee la memoria almacenándose en el *mdr* el contenido de la posición indicada por el *mar*
  - se transfiere el contenido del *mdr* al *ir*
  - se incrementará el *pc* en 1 para que quede apuntando a la instrucción siguiente

  Decodificación-ejecución.
  - Específica para cada instrucción



VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo de Comportamiento (3)

Transferencias elementales para cada instrucción

LDA	**STA**	**SUM**
mar <-- <ir> lectura ac <-- <mdr> n <-- <ac(11)> IF ac="000000000000" THEN z <-- '1' ELSE z <-- '0'	mdr <-- <ac> mar <-- <ir> escritura	mar <-- <ir> lectura ac <-- <ac>+<mdr> n <-- <ac(11)> IF ac="000000000000" THEN z <-- '1' ELSE z <-- '0
LDAI	**SUMI**	**NOR**
ac <-- <ir> n <-- <ac(11)> IF ac="000000000000" THEN z <-- '1' ELSE z <-- '0'	mdr <-- <ir> ac <-- <ac>+<mdr> n <-- <ac(11)> IF ac="000000000000" THEN z <-- '1' ELSE z <-- '0'	mar <-- <ir> lectura ac <-- <mdr> n <-- <ac(11)> IF ac="000000000000" THEN z <-- '1' ELSE z <-- '0'
JZ	**JNZ**	**JN**
IF nz(0) = '1' THEN pc <-- <ir>	IF nz(0) = '0' THEN pc <-- <ir>	IF nz(1) = '1' THEN pc <-- <ir>
JNN	**HALT**	
IF nz(1) = '0' THEN pc <-- <ir>	stop <-- '1'	

VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo de Comportamiento (4)

Visualización del resultado de la ejecución

- Para trazar programas máquina será necesario que el modelo permita visualizar el contenido de registros y memoria.
- Como ambos están soportados por variables cuyos valores sólo son accesibles dentro del proceso, su visualización debe hacerse cuando finalice la ejecución del programa máquina, antes de que el proceso pase al estado de espera.
- Visualizaremos los registros con el siguiente formato de línea:

<PC> = contenido del *pc* <AC> = contenido del *ac* <NZ> = contenido de *nz*

```
VARIABLE l : LINE;
```

```
CONSTANT c   : STRING :=  " = ";
CONSTANT cac : STRING :=  "  <AC> = ";
CONSTANT cpc : STRING :=  "  <PC> = ";
CONSTANT cnz : STRING :=  "  <NZ> = ";
```

```
WRITE(l,cpc);          -- rotulo delpc, es decir, <PC> =
WRITE(l, pc);          -- valor de pc
WRITE(l,cac);          -- rótulo del ac, es decir, <AC> =
WRITE(l, ac);          -- valor de ac
WRITE(l,cnz);          -- rótulo del nz, es decir, <NZ> =
WRITE(l,nz);           -- valor de nz
```

visualización de registros

```
WRITELINE(OUTPUT,l);
```

```
FOR i IN 0 TO 29 LOOP
WRITE(l,i);            -- valor de la dirección
WRITE(l, c);           -- signo =
WRITE(l, mem(i));      -- valor del contenido de memoria
WRITELINE(OUTPUT, l);
END LOOP;
```

visualización de memoria

VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo de Comportamiento (5)

Código VHDL del modelo completo (1)

```
USE STD.TEXTIO.ALL;
USE WORK.utilidad.ALL;
ENTITY computador IS
   PORT (start, reset : IN bit);
END computador;
ARCHITECTURE comportamiento OF computador IS
BEGIN
  PROCESS(start, reset)
    TYPE array_memoria IS
    ARRAY (NATURAL RANGE 0 TO 255) OF bit_vector(11 DOWNTO 0);
    VARIABLE mem : array_memoria :=
-- programa a ejecutar
("000100000100", --  0 LDA (STA indice)
 "010100000001", --  1 SUMI 1
 "001000000100", --  2 STA (STA indice+1)
 "000100001110", --  3 LDA indice
 "001000010011", --  4 STA indice
 "010100000001", --  5 SUMI 1
 "001000001110", --  6 STA indice
 "000100001101", --  7 LDA limite
 "011000001101", --  8 NOR limite
 "010100000001", --  9 SUMI 1
 "001100001110", -- 10 SUM indice
 "100000000000", -- 11 JNZ 0
 "000000000000", -- 12 HALT
 "000000011110", -- 13 limite
 "000000010100", -- 14 indice
  OTHERS => "000000000000");
    VARIABLE l : LINE;
    CONSTANT c : STRING := " = ";
    CONSTANT cac : STRING := "  <AC> = ";
    CONSTANT cpc : STRING := "<PC> = ";
    CONSTANT cnz : STRING := "  <NZ> = ";
    VARIABLE mdr, mar, pc, ir, ac : bit_vector(11 DOWNTO 0);
    VARIABLE nz : bit_vector(1 DOWNTO 0);
    VARIABLE stop : bit;
```

Modelo de Comportamiento (6)

Código VHDL del modelo completo (2)

```
BEGIN
-- inicialización de la máquina (reset)
   IF reset'EVENT AND reset = '1'
   THEN
      pc := "000000000000";
-- operación de la máquina
   ELSIF start'EVENT AND START = '1'
   THEN
      WHILE stop = '0' LOOP
-- ciclo de búsqueda de la siguiente instrucción
      mar := pc;                                  -- mar <-- <pc>
      mdr := mem(b_a_n(mar));                     -- lectura
      ir := mdr;                                  -- ir  <-- <mdr>
      pc := n_a_b((b_a_n(pc) + 1), 12);           -- pc  <-- <pc> + 1
                  -- ciclo de ejecución
      CASE ir(11 DOWNTO 8) IS
         WHEN "0001" =>                           -- LDA
            mar := "0000" & ir(7 DOWNTO 0);       -- mar <-- <ir>
            mdr := mem(b_a_n(mar));               -- lectura
            ac := mdr;                            -- ac <-- <mdr>
            nz(1) := ac(11);                      -- n <-- <ac(11)>
            IF ac = "000000000000"                -- IF ac="000000000000"
               THEN nz(0) := '1';                 --  THEN z <-- '1'
               ELSE nz(0) := '0';                 --  ELSE z <-- '0'
            END IF;
         WHEN "0010" =>                           -- STA
            mdr := ac;                            -- mdr <-- <ac>
            mar := "0000" & ir(7 DOWNTO 0);       -- mar <-- <ir>
            mem(b_a_n(mar)) := mdr;               -- escritura
         WHEN "0011" =>                           -- SUM
            mar := "0000" & ir(7 DOWNTO 0);       -- mar <-- <ir>
            mdr := mem(b_a_n(mar));               -- lectura
      ac := n_a_b((b_a_n(ac) + b_a_n(mdr)), 12); -- ac <-- <ac>+<mdr>
            nz(1) := ac(11);                      -- n <-- <ac(11)>
            IF ac = "000000000000"                -- IF ac="000000000000"
               THEN nz(0) := '1';                 --  THEN z <-- '1'
               ELSE nz(0) := '0';                 --  ELSE z <-- '0'
            END IF;
```

Modelo de Comportamiento (7)

Código VHDL del modelo completo (3)

```
WHEN "0100" =>                                  -- LDAI
        ac := "0000" & ir(7 DOWNTO 0);          -- ac <-- <ir>
        nz(1) := ac(11);                        -- n <-- <ac(11)>
        IF ac = "000000000000"                  -- IF ac="000000000000"
           THEN nz(0) := '1';                   --  THEN z <-- '1'
           ELSE nz(0) := '0';                   --  ELSE z <-- '0'
        END IF;
      WHEN "0101" =>                            -- SUMI
        mdr := "0000" & ir(7 DOWNTO 0);         -- mdr <-- <ir>
   ac := n_a_b((b_a_n(ac) + b_a_n(mdr)), 12);   -- ac <-- <ac>+<mdr>
        nz(1) := ac(11);                        -- n <-- <ac(11)>
        IF ac = "000000000000"                  -- IF ac="00000000000"
           THEN nz(0) := '1';                   --  THEN z <-- '1'
           ELSE nz(0) := '0';                   --  ELSE z <-- '0'
        END IF;
      WHEN "0110" =>                            -- NOR
        mar := "0000" & ir(7 DOWNTO 0);         -- mar <-- <ir>
        mdr := mem(b_a_n(mar));                 -- lectura
        ac := ac NOR mdr;                       -- ac <-- <mdr>
        nz(1) := ac(11);                        -- n <-- <ac(11)>
        IF ac = "000000000000"                  -- IF ac="000000000000"
           THEN nz(0) := '1';                   --  THEN z <-- '1'
           ELSE nz(0) := '0';                   --  ELSE z <-- '0'
        END IF;
      WHEN "0111" =>                            -- JZ
        IF nz(0) = '1' THEN
           pc := "0000" & ir(7 DOWNTO 0);       -- pc <-- <ir>
        END IF;
      WHEN "1000" =>                            -- JNZ
        IF nz(0) = '0' THEN
           pc := "0000" & ir(7 DOWNTO 0);       -- pc <-- <ir>
        END IF;
              -- STOP
        stop := '1';
```

```
 WHEN "1001" =>                                 -- JN
        IF nz(1) = '1' THEN
           pc := "0000" & ir(7 DOWNTO 0);       -- pc <-- <ir>
        END IF;
      WHEN "1010" =>                            -- JNN
        IF nz(1) = '0' THEN
           pc := "0000" & ir(7 DOWNTO 0);       -- pc <-- <ir>
        END IF;
      WHEN OTHERS =>
END CASE;
  -- visualiza pc, ac y nz
      WRITE(l,cpc);
      WRITE(l, pc);
      WRITE(l,cac);
      WRITE(l, ac);
      WRITE(l,cnz);
      WRITE(l,nz);
      WRITELINE(OUTPUT,l);
   END LOOP;
  -- visualiza las  primeras posiciones de memoria
     FOR i IN 0 TO 30 LOOP  --
      WRITE(l,i);                --
      WRITE(l, c);               --
      WRITE(l, mem(i));          --
      WRITELINE(OUTPUT, l); --
     END LOOP;                  --
  END IF;
 END PROCESS;
END comportamiento;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo de Comportamiento (8)

Código VHDL del paquete *utilidad*)

El paquete contiene las funciones *b_a_n(bits : IN bit_vector)* y *n_a_b(nat, nb : IN NATURAL)* que realizan la conversión de tipo BIT_VECTOR a tipo NATURAL, y de tipo NATURAL a tipo BIT_VECTOR, respectivamente.

El segundo argumento de esta última función (*nb*) determina el número de bits del resultado.

```
PACKAGE utilidad IS
  FUNCTION b_a_n (bits : IN bit_vector) RETURN natural;
  FUNCTION n_a_b (nat, nb : IN NATURAL) RETURN bit_vector;
END utilidad;

PACKAGE BODY utilidad IS
  FUNCTION b_a_n (bits : IN bit_vector) RETURN natural IS
    VARIABLE result : natural := 0;
  BEGIN
    FOR indice IN bits'RANGE LOOP
      result := result * 2 + bit'POS(bits(indice));
    END LOOP;
    RETURN result;
  END b_a_n;
  FUNCTION n_a_b (nat, nb : IN NATURAL) RETURN bit_vector IS
  VARIABLE nat_var : bit_vector(nb - 1 DOWNTO 0);
  VARIABLE temp1, temp2 : NATURAL := 0;
BEGIN
  temp1 := nat;
  FOR i IN nb-1 DOWNTO 0 LOOP
    temp2 := temp1/(2**i);
    temp1 := temp1 mod (2**i);
    IF ( temp2 = 1 ) THEN nat_var(i) := '1';
                     ELSE nat_var(i) := '0';
    END IF;
  END LOOP;
  RETURN nat_var;
END n_a_b;
END utilidad;
```

Modelo de Comportamiento (9)

Resultados de la simulación

Registros

```
# <PC> = 000000000001  <AC> = 001000010011  <NZ> = 00
# <PC> = 000000000010  <AC> = 001000010100  <NZ> = 00
# <PC> = 000000000011  <AC> = 001000010100  <NZ> = 00
# <PC> = 000000000100  <AC> = 000000010100  <NZ> = 00
# <PC> = 000000000101  <AC> = 000000010100  <NZ> = 00
# <PC> = 000000000110  <AC> = 000000010101  <NZ> = 00
# <PC> = 000000000111  <AC> = 000000010101  <NZ> = 00
# <PC> = 000000001000  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001001  <AC> = 111111100001  <NZ> = 10
# <PC> = 000000001010  <AC> = 111111100010  <NZ> = 10
# <PC> = 000000001011  <AC> = 111111110111  <NZ> = 10
# <PC> = 000000000000  <AC> = 111111110111  <NZ> = 00
# <PC> = 000000000001  <AC> = 001000010100  <NZ> = 00
# <PC> = 000000000010  <AC> = 001000010101  <NZ> = 00
# <PC> = 000000000011  <AC> = 001000010101  <NZ> = 00
# <PC> = 000000000100  <AC> = 000000010101  <NZ> = 00
# <PC> = 000000000101  <AC> = 000000010101  <NZ> = 00
# <PC> = 000000000110  <AC> = 000000010110  <NZ> = 00
# <PC> = 000000000111  <AC> = 000000010110  <NZ> = 00
# <PC> = 000000001000  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001001  <AC> = 111111100001  <NZ> = 10
# <PC> = 000000001010  <AC> = 111111100010  <NZ> = 10
# <PC> = 000000001011  <AC> = 111111111000  <NZ> = 10
# <PC> = 000000000000  <AC> = 111111111000  <NZ> = 10
# <PC> = 000000000001  <AC> = 001000010101  <NZ> = 00
# <PC> = 000000000010  <AC> = 001000010110  <NZ> = 00
# <PC> = 000000000011  <AC> = 001000010110  <NZ> = 00
# <PC> = 000000000100  <AC> = 000000010110  <NZ> = 00
# <PC> = 000000000101  <AC> = 000000010110  <NZ> = 00
# <PC> = 000000000110  <AC> = 000000010111  <NZ> = 00
# <PC> = 000000000111  <AC> = 000000010111  <NZ> = 00
# <PC> = 000000001000  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001001  <AC> = 111111100001  <NZ> = 10
# <PC> = 000000001010  <AC> = 111111100010  <NZ> = 10
# <PC> = 000000001011  <AC> = 111111111001  <NZ> = 10
# <PC> = 000000000000  <AC> = 111111111001  <NZ> = 10
# <PC> = 000000000001  <AC> = 001000010110  <NZ> = 00
# <PC> = 000000000010  <AC> = 001000010111  <NZ> = 00
# <PC> = 000000000011  <AC> = 001000010111  <NZ> = 00
# <PC> = 000000000100  <AC> = 000000010111  <NZ> = 00
# <PC> = 000000000101  <AC> = 000000010111  <NZ> = 00
# <PC> = 000000000110  <AC> = 000000011000  <NZ> = 00
# <PC> = 000000000111  <AC> = 000000011000  <NZ> = 00
# <PC> = 000000001000  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001001  <AC> = 111111100001  <NZ> = 10
# <PC> = 000000001010  <AC> = 111111100010  <NZ> = 10
# <PC> = 000000001011  <AC> = 111111111010  <NZ> = 10
# <PC> = 000000000000  <AC> = 111111111010  <NZ> = 10
# <PC> = 000000000001  <AC> = 001000010111  <NZ> = 00
# <PC> = 000000000010  <AC> = 001000011000  <NZ> = 00
# <PC> = 000000000011  <AC> = 001000011000  <NZ> = 00
# <PC> = 000000000100  <AC> = 000000011000  <NZ> = 00
# <PC> = 000000000101  <AC> = 000000011000  <NZ> = 00
# <PC> = 000000000110  <AC> = 000000011001  <NZ> = 00
# <PC> = 000000000111  <AC> = 000000011001  <NZ> = 00
# <PC> = 000000001000  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001001  <AC> = 111111100001  <NZ> = 10
# <PC> = 000000001010  <AC> = 111111100010  <NZ> = 10
# <PC> = 000000001011  <AC> = 111111111011  <NZ> = 10
# <PC> = 000000000000  <AC> = 111111111011  <NZ> = 10
```

```
# <PC> = 000000000001  <AC> = 001000011000  <NZ> = 00
# <PC> = 000000000010  <AC> = 001000011001  <NZ> = 00
# <PC> = 000000000011  <AC> = 001000011001  <NZ> = 00
# <PC> = 000000000100  <AC> = 000000011001  <NZ> = 00
# <PC> = 000000000101  <AC> = 000000011001  <NZ> = 00
# <PC> = 000000000110  <AC> = 000000011010  <NZ> = 00
# <PC> = 000000000111  <AC> = 000000011010  <NZ> = 00
# <PC> = 000000001000  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001001  <AC> = 111111100001  <NZ> = 10
# <PC> = 000000001010  <AC> = 111111100010  <NZ> = 10
# <PC> = 000000001011  <AC> = 111111111100  <NZ> = 10
# <PC> = 000000000000  <AC> = 111111111100  <NZ> = 10
# <PC> = 000000000001  <AC> = 001000011001  <NZ> = 00
# <PC> = 000000000010  <AC> = 001000011010  <NZ> = 00
# <PC> = 000000000011  <AC> = 001000011010  <NZ> = 00
# <PC> = 000000000100  <AC> = 000000011010  <NZ> = 00
# <PC> = 000000000101  <AC> = 000000011010  <NZ> = 00
# <PC> = 000000000110  <AC> = 000000011011  <NZ> = 00
# <PC> = 000000000111  <AC> = 000000011011  <NZ> = 00
# <PC> = 000000001000  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001001  <AC> = 111111100001  <NZ> = 10
# <PC> = 000000001010  <AC> = 111111100010  <NZ> = 10
# <PC> = 000000001011  <AC> = 111111111101  <NZ> = 10
# <PC> = 000000000000  <AC> = 111111111101  <NZ> = 10
# <PC> = 000000000001  <AC> = 001000011010  <NZ> = 00
# <PC> = 000000000010  <AC> = 001000011011  <NZ> = 00
# <PC> = 000000000011  <AC> = 001000011011  <NZ> = 00
# <PC> = 000000000100  <AC> = 000000011011  <NZ> = 00
# <PC> = 000000000101  <AC> = 000000011011  <NZ> = 00
# <PC> = 000000000110  <AC> = 000000011100  <NZ> = 00
# <PC> = 000000000111  <AC> = 000000011100  <NZ> = 00
# <PC> = 000000001000  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001001  <AC> = 111111100001  <NZ> = 10
# <PC> = 000000001010  <AC> = 111111100010  <NZ> = 10
# <PC> = 000000001011  <AC> = 111111111110  <NZ> = 10
# <PC> = 000000000000  <AC> = 111111111110  <NZ> = 10
# <PC> = 000000000001  <AC> = 001000011011  <NZ> = 00
# <PC> = 000000000010  <AC> = 001000011100  <NZ> = 00
# <PC> = 000000000011  <AC> = 001000011100  <NZ> = 00
# <PC> = 000000000100  <AC> = 000000011100  <NZ> = 00
# <PC> = 000000000101  <AC> = 000000011100  <NZ> = 00
# <PC> = 000000000110  <AC> = 000000011101  <NZ> = 00
# <PC> = 000000000111  <AC> = 000000011101  <NZ> = 00
# <PC> = 000000001000  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001001  <AC> = 111111100001  <NZ> = 10
# <PC> = 000000001010  <AC> = 111111100010  <NZ> = 10
# <PC> = 000000001011  <AC> = 111111111111  <NZ> = 10
# <PC> = 000000000000  <AC> = 111111111111  <NZ> = 10
# <PC> = 000000000001  <AC> = 001000011100  <NZ> = 00
# <PC> = 000000000010  <AC> = 001000011101  <NZ> = 00
# <PC> = 000000000011  <AC> = 001000011101  <NZ> = 00
# <PC> = 000000000100  <AC> = 000000011101  <NZ> = 00
# <PC> = 000000000101  <AC> = 000000011101  <NZ> = 00
# <PC> = 000000000110  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000000111  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001000  <AC> = 000000011110  <NZ> = 00
# <PC> = 000000001001  <AC> = 111111100001  <NZ> = 10
# <PC> = 000000001010  <AC> = 111111100010  <NZ> = 10
# <PC> = 000000001011  <AC> = 000000000000  <NZ> = 01
# <PC> = 000000001100  <AC> = 000000000000  <NZ> = 01
```

Memoria

```
# 00 = 000100000100
# 01 = 010100000001
# 02 = 001000000100
# 03 = 000100001110
# 04 = 001000011101
# 05 = 010100000001
# 06 = 001000001110
# 07 = 000100001101
# 08 = 011000001101
# 09 = 010100000001
# 10 = 001100001110
# 11 = 100000000000
# 12 = 000000000000
# 13 = 000000011110
# 14 = 000000011110
# 15 = 000000000000
# 16 = 000000000000
# 17 = 000000000000
# 18 = 000000000000
# 19 = 000000000000
# 20 = 000000010100
# 21 = 000000010101
# 22 = 000000010110
# 23 = 000000010111
# 24 = 000000011000
# 25 = 000000011001
# 26 = 000000011010
# 27 = 000000011011
# 28 = 000000011100
# 29 = 000000011101
```

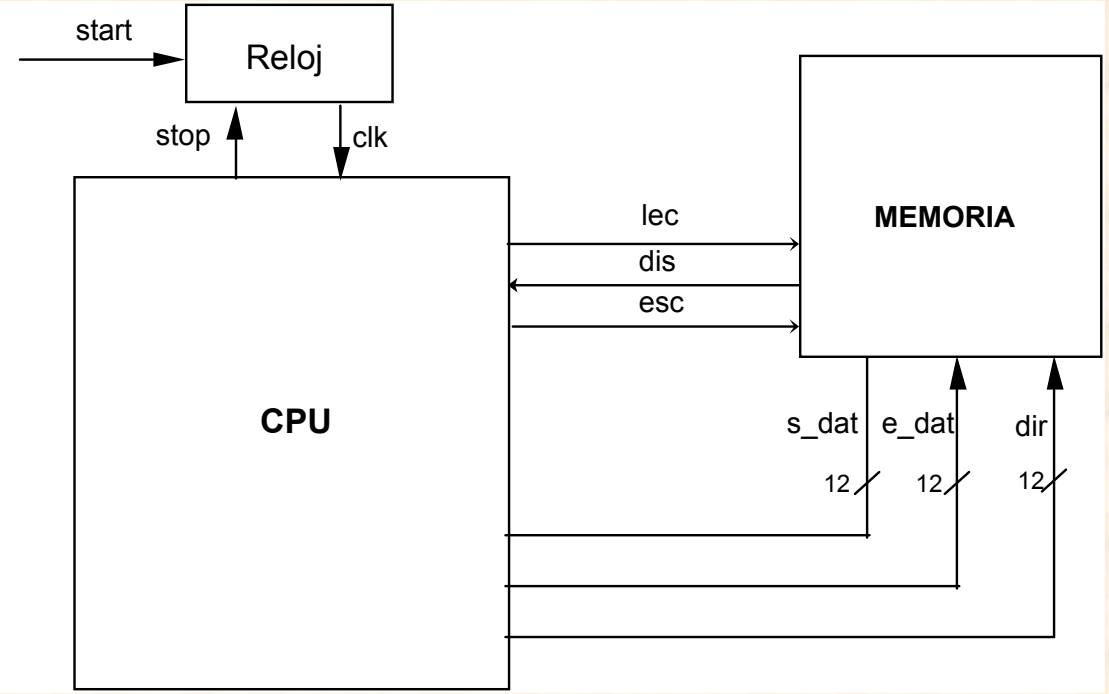
Modelo CPU-Memoria (1)

- En este segundo modelo descompondremos el computador en sus dos unidades básica:

  ***UCP* y la *Memoria*.**

- Para cada unidad diseñaremos una entidad que describirá su comportamiento funcional y temporal.

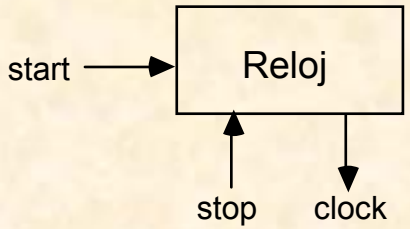
- El *reloj* del sistema también lo modelaremos como una entidad

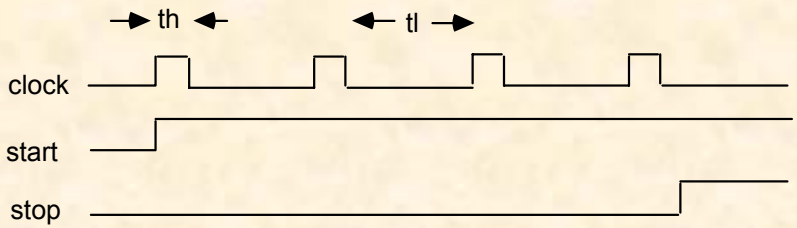


Modelo CPU-Memoria (2)

Modelo VHDL del Reloj

- Consta de dos entradas de control:
  - *start* para iniciar la generación de pulsos
  - *stop* para detenerla.
- Ambas se activan con flancos positivos de sus valores.
- El tiempo de permanencia en alta y baja de la señal de salida *clock* viene determinado por los parámetros genéricos *th* y *tl* respectivamente






```
ENTITY reloj IS
      GENERIC(tl : TIME := 20 ns; th : TIME := 5 ns);
      PORT(start, stop : IN bit; clock : OUT bit);
END reloj;

  ARCHITECTURE comportamiento OF reloj IS
  SIGNAL clk : bit := '0';
  BEGIN
    PROCESS(start, stop, clk)
    VARIABLE clke : bit := '0';
    BEGIN
      IF (start = '1' and NOT start'STABLE) THEN
      clke := '1';
      clk <= TRANSPORT '1' AFTER 0 ns;
      clk <= TRANSPORT '0' AFTER th;
      END IF;
      IF (stop = '1' and NOT stop'STABLE) THEN
      clke := '0';
      clk <= TRANSPORT '0' AFTER 0 ns;
      END IF;
      IF (clk = '0' and NOT clk'STABLE and clke = '1') THEN
      clk <= TRANSPORT '1' AFTER tl;
      clk <= TRANSPORT '0' AFTER tl+th;
      END IF;
      clock <= clk;
    END PROCESS;
  END comportamiento;
```

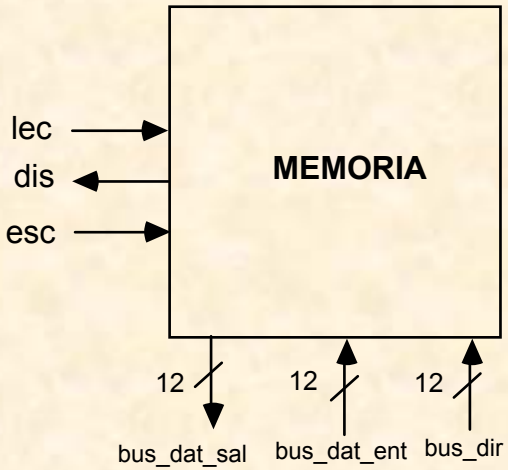

VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

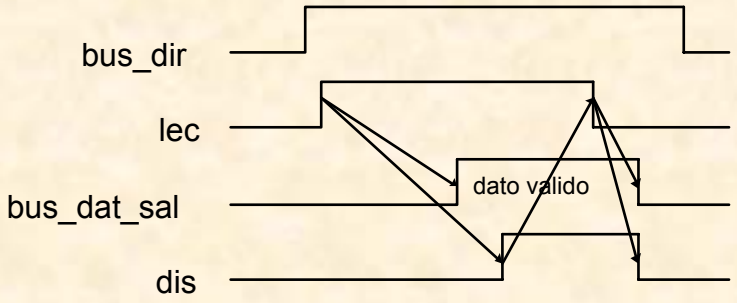
Modelo CPU-Memoria (3)

Modelo VHDL de la Memoria (1)

Memoria asíncrona con buses independientes de entrada y salida de datos: *bus_dat_sal* y *bus_dat_ent*..
El control se realiza con tres señales binarias: *lec*, *esc* y *dis*.
La primera para leer, la segunda para escribir, y la tercera para señalar la finalización de la operación realizada.
Los buses de datos y el de direcciones (bus_dir) son todos de 12 bits de longitud

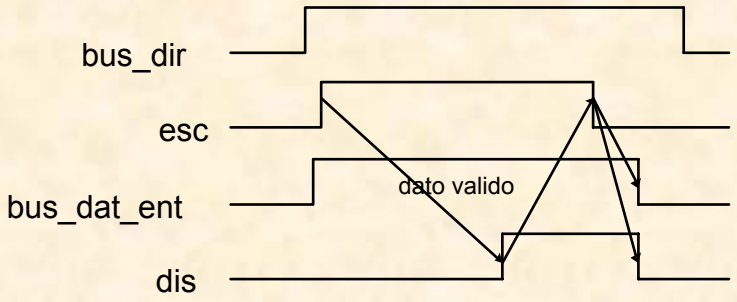
Las operaciones de lectura y escritura se realizan asíncronamente con un protocolo de señales tipo *hand-shaking*





lectura de memoria

```
ENTITY memoria IS
  PORT (bus_dat_sal : OUT bit_vector(11 DOWNTO 0);
        bus_dat_ent :  IN bit_vector(11 DOWNTO 0);
        bus_dir : IN bit_vector(11 DOWNTO 0);
        lec, esc : IN bit;
        dis : OUT bit);
END memoria;
```



escritura en memoria

Modelo CPU-Memoria (4)

Modelo VHDL de la Memoria (2)

```
ARCHITECTURE comportamiento OF memoria IS
BEGIN
  PROCESS
    TYPE array_memoria IS
    ARRAY (NATURAL RANGE 0 TO 255) OF bit_vector(11
DOWNTO 0);
    VARIABLE mem : array_memoria :=
                  ("000100000100", --  0 LDA (STA índice)
                   "010100000001", --  1 SUMI 1
                   "001000000100", --  2 STA (STA índice+1)
                   "000100001110", --  3 LDA índice
                   "001000010011", --  4 STA índice
                   "010100000001", --  5 SUMI 1
                   "001000001110", --  6 STA índice
                   "000100001101", --  7 LDA limite
                   "011000001101", --  8 NOR limite
                   "010100000001", --  9 SUMI 1
                   "001100001110", -- 10 SUM índice
                   "100000000000", -- 11 JNZ 0
                   "000000000000", -- 12 HALT
                   "000000011110", -- 13 limite
                   "000000010100", -- 14 índice
                    OTHERS => "000000000000");
    SUBTYPE dir IS NATURAL RANGE 0 TO 255;
    VARIABLE direc : dir;
    VARIABLE l : LINE;
    CONSTANT c : STRING := " = ";
```

```
BEGIN
    bus_dat_sal <= "000000000000" AFTER 5 ns;
    dis <= '0'AFTER 5 ns;
    WAIT UNTIL (lec = '1') OR (esc = '1');
    direc := b_a_n(bus_dir);
       -- visualiza las primeras posiciones de memoria
       IF (esc = '1') AND (lec = '1')  THEN                --
          FOR i IN 0 TO 30 LOOP                            --
             WRITE(l,i);                                   --
             WRITE(l, c);                                  --
             WRITE(l, mem(i));
                 --
             WRITELINE(OUTPUT, l);                         --
          END LOOP;
                 --
       -- cuando se activan simultnemente  lec  y  esc
       ELSIF esc = '1' THEN
          mem(direc) := bus_dat_ent;
          dis <= '1' AFTER 20 ns;
          WAIT UNTIL esc = '0';
       ELSE
          bus_dat_sal <= mem(direc) AFTER 15 ns;
          dis <= '1' AFTER 20 ns;
          WAIT UNTIL lec = '0';
       END IF;
  END PROCESS;
END comportamiento;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

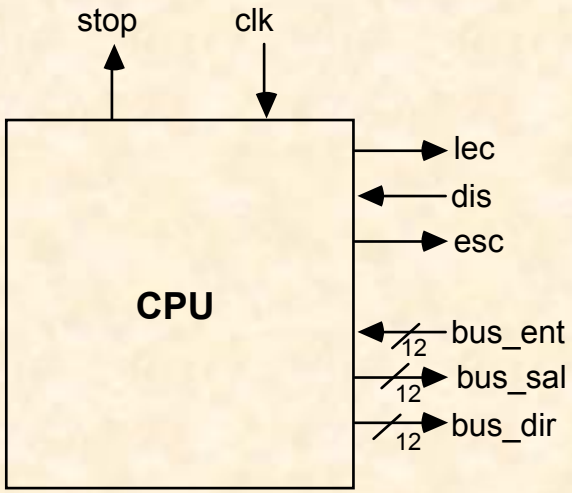
Modelo CPU-Memoria (5)

Modelo VHDL de la Unidad Central de Proceso

Su periferia dispondrá de los buses de datos direcciones y control necesarios para comunicarse con la memoria.

Se comunicará con el reloj a través de dos señales binarias:

- *clk* para recibir la señal de reloj
- *stop* para detener la generación de pulsos cuando se ejecute la instrucción de parada HALT

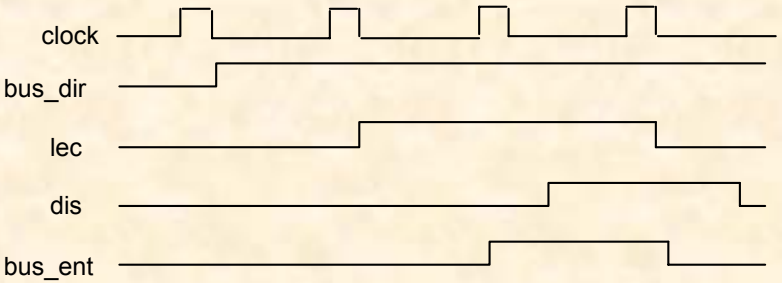



```
ENTITY cpu IS
  PORT (bus_sal : OUT bit_vector(11 DOWNTO 0);
        bus_ent :  IN bit_vector(11 DOWNTO 0);
        bus_dir : OUT bit_vector(11 DOWNTO 0);
        lec, esc, stop : OUT bit;
        dis, clk : IN bit);
END cpu;
```

Modelo CPU-Memoria (6)

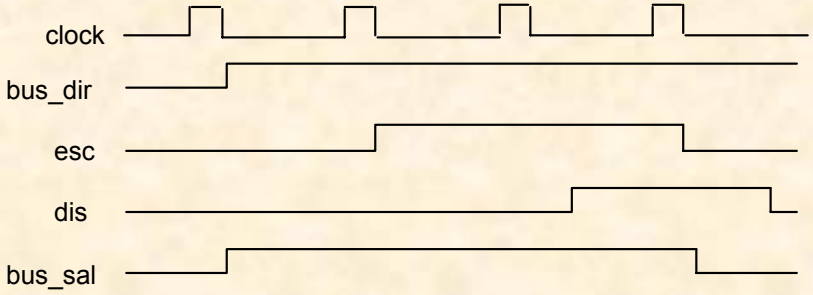
Procedimiento de lectura

```
PROCEDURE lectura IS
   BEGIN
      WAIT UNTIL (clk'EVENT AND clk = '0');
      bus_dir <= mar;
      WAIT UNTIL (clk'EVENT AND clk = '0');
      lec <= '1';
      WAIT UNTIL dis = '1';
      WAIT UNTIL (clk'EVENT AND clk = '0');
      mdr := bus_ent;
      WAIT UNTIL (clk'EVENT AND clk = '0');
      lec <= '0';
   END lectura;
```



Procedimiento de escritura

```
PROCEDURE escritura IS
   BEGIN
      WAIT UNTIL (clk'EVENT AND clk = '0');
      bus_dir  <= mar;
      bus_sal  <= mdr;
      WAIT UNTIL (clk'EVENT AND clk = '0');
      esc <= '1';
      WAIT UNTIL dis = '1';
      WAIT UNTIL (clk'EVENT AND clk = '0');
      esc <= '0';
   END escritura;
```



VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo CPU-Memoria (7)

```
USE WORK.utilidad.ALL;
ENTITY cpu IS
  PORT (bus_sal : OUT bit_vector(11 DOWNTO 0);
        bus_ent :  IN bit_vector(11 DOWNTO 0);
        bus_dir : OUT bit_vector(11 DOWNTO 0);
        lec, esc, stop : OUT bit;
        dis, clk : IN bit);
END cpu;
ARCHITECTURE comportamiento OF cpu IS
BEGIN
  PROCESS
    VARIABLE mdr, mar, pc, ir, ac : bit_vector(11 DOWNTO 0);
    VARIABLE nz : bit_vector(1 DOWNTO 0);
    PROCEDURE lectura IS
    BEGIN
       WAIT UNTIL (clk'EVENT AND clk = '0');
       bus_dir <= mar;
       WAIT UNTIL (clk'EVENT AND clk = '0');
       lec <= '1';
       WAIT UNTIL dis = '1';
       WAIT UNTIL (clk'EVENT AND clk = '0');
       mdr := bus_ent;
       WAIT UNTIL (clk'EVENT AND clk = '0');
       lec <= '0';
    END lectura;
    PROCEDURE escritura IS
    BEGIN
       WAIT UNTIL (clk'EVENT AND clk = '0');
       bus_dir  <= mar;
       bus_sal  <= mdr;
       WAIT UNTIL (clk'EVENT AND clk = '0');
       esc <= '1';
       WAIT UNTIL dis = '1';
       WAIT UNTIL (clk'EVENT AND clk = '0');
       esc <= '0';
    END escritura;
```

Código VHDL completo de la Unidad Central de Proceso (1)

```
BEGIN
      mar := pc;                                  -- mar <-- <pc>
      lectura;                                    -- lectura
      ir := mdr;                                  -- ir  <-- <mdr>
      pc := n_a_b((b_a_n(pc) + 1), 12);           -- pc  <-- <pc> + 1
      WAIT UNTIL (clk'EVENT AND clk = '0');       -- decodificacion
```

Modelo CPU-Memoria (8)

```
CASE ir(11 DOWNTO 8) IS
        WHEN "0001" =>                                  -- LDA
           mar := "0000" & ir(7 DOWNTO 0);       -- mar <-- <ir>
           lectura;                                     -- lectura
           ac := mdr;                                   -- ac <-- <mdr>
           nz(1) := ac(11);                             -- n <-- <ac(11)>
           IF ac = "000000000000"                       -- IF ac="000000000000"
              THEN nz(0) := '1';                        --  THEN z <-- '1'
              ELSE nz(0) := '0';                        --  ELSE z <-- '0'
           END IF;
        WHEN "0010" =>                                  -- STA
           mdr := ac;                                   -- mdr <-- <ac>
           mar := "0000" & ir(7 DOWNTO 0);              -- mar <-- <ir>
           escritura;                                   -- escritura
        WHEN "0011" =>                                  -- SUM
           mar := "0000" & ir(7 DOWNTO 0);              -- mar <-- <ir>
           lectura;                          -- lectura
     ac := n_a_b((b_a_n(ac) + b_a_n(mdr)), 12);         -- ac <-- <ac>+<mdr>
           nz(1) := ac(11);                             -- n <-- <ac(11)>
           IF ac = "000000000000"                       -- IF ac="000000000000"
              THEN nz(0) := '1';                        --  THEN z <-- '1'
              ELSE nz(0) := '0';                        --  ELSE z <-- '0'
           END IF;
        WHEN "0100" =>                                  -- LDAI
           ac := "0000" & ir(7 DOWNTO 0);               -- ac <-- <ir>
           nz(1) := ac(11);                             -- n <-- <ac(11)>
           IF ac = "000000000000"                       -- IF ac="000000000000"
              THEN nz(0) := '1';                        --  THEN z <-- '1'
              ELSE nz(0) := '0';                        --  ELSE z <-- '0'
           END IF;
        WHEN "0101" =>                                  -- SUMI
           mdr := "0000" & ir(7 DOWNTO 0);              -- mdr <-- <ir>
     ac := n_a_b((b_a_n(ac) + b_a_n(mdr)), 12);         -- ac <-- <ac>+<mdr>
           nz(1) := ac(11);                             -- n <-- <ac(11)>
           IF ac = "000000000000"                       -- IF ac="000000000000"
              THEN nz(0) := '1';
                   --  THEN z <-- '1'
              ELSE nz(0) := '0';
                   --  ELSE z <-- '0'
           END IF;
```

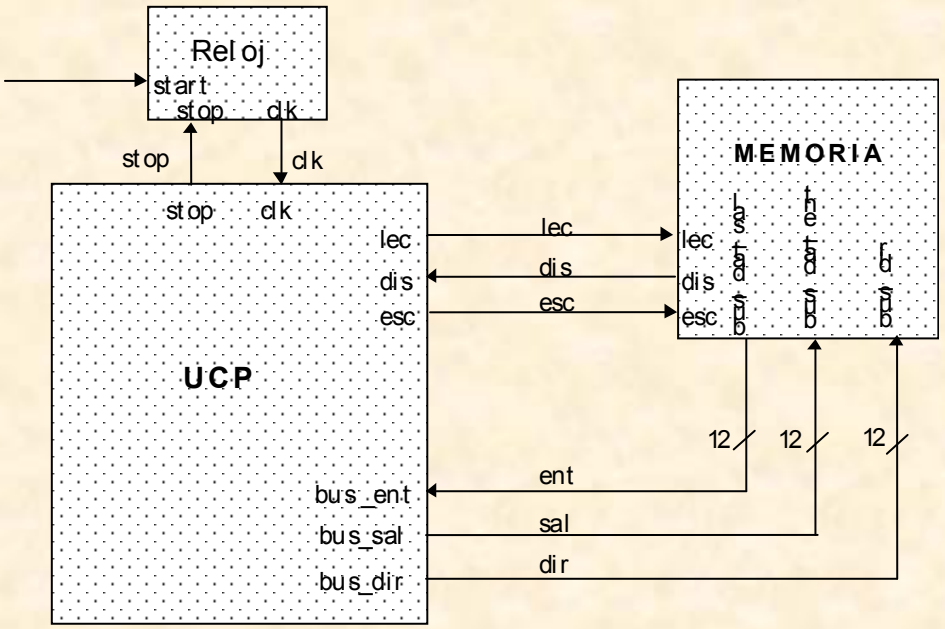
Código VHDL completo de la Unidad Central de Proceso (2)

```
WHEN "0110" =>                          -- NOR
           mar := "0000" & ir(7 DOWNTO 0);      -- mar <-- <ir>
           lectura;                             -- lectura
           ac := ac NOR mdr;                    -- ac <-- <mdr>
           nz(1) := ac(11);                     -- n <-- <ac(11)>
           IF ac = "000000000000"               -- IF ac="000000000000"
              THEN nz(0) := '1';                --  THEN z <-- '1'
              ELSE nz(0) := '0';                --  ELSE z <-- '0'
           END IF
        WHEN "0111" =>                          -- JZ
           IF nz(0) = '1' THEN
              pc := "0000" & ir(7 DOWNTO 0);    -- pc <-- <ir>
           END IF;
        WHEN "1000" =>                                  -- JNZ
           IF nz(0) = '0' THEN
              pc := "0000" & ir(7 DOWNTO 0);            -- pc <-- <ir>
           END IF;
        WHEN "1001" =>                                  -- JN
           IF nz(1) = '1' THEN
              pc := "0000" & ir(7 DOWNTO 0);            -- pc <-- <ir>
           END IF;
        WHEN "1010" =>                                  -- JNN
           IF nz(1) = '0' THEN
              pc := "0000" & ir(7 DOWNTO 0);            -- pc <-- <ir>
           END IF;
        WHEN OTHERS =>
           lec <= '1';
           esc <= '1';                          -- volcado memoria
           stop <= '1', '0' AFTER 5 ns;         -- parada
     END CASE;
  END PROCESS;
END comportamiento;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo CPU-Memoria (9)

Modelo Estructural CPU-Memoria




```
ENTITY comput IS    PORT (start : IN bit);
END comput;
ARCHITECTURE estructura OF comput IS
COMPONENT cpu
  PORT (bus_sal : OUT bit_vector(11 DOWNTO 0);
        bus_ent :  IN bit_vector(11 DOWNTO 0);
        bus_dir : OUT bit_vector(11 DOWNTO 0);
        lec, esc, stop : OUT bit;
        dis, clk : IN bit);
END COMPONENT;
COMPONENT memoria
  PORT (bus_dat_sal : OUT bit_vector(11 DOWNTO 0);
        bus_dat_ent :  IN bit_vector(11 DOWNTO 0);
        bus_dir : IN bit_vector(11 DOWNTO 0);
        lec, esc : IN bit;
        dis : OUT bit);
END COMPONENT;
COMPONENT reloj
        GENERIC(tl : TIME := 20 ns; th : TIME := 5 ns);
        PORT(start, stop : IN bit;
             clock : OUT bit);
END COMPONENT;
SIGNAL sal, ent, dir : bit_vector(11 DOWNTO 0);
SIGNAL lec, dis, esc, stop, clk : bit;
BEGIN
    clock : reloj GENERIC MAP(20 ns, 5 ns)
PORT MAP(start, stop, clk);
    memor : memoria PORT MAP(ent, sal, dir, lec, esc, dis);

    unice : cpu PORT MAP(sal, ent, dir, lec, esc, stop, dis, clk);
END estructura;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo CPU-Memoria (10)

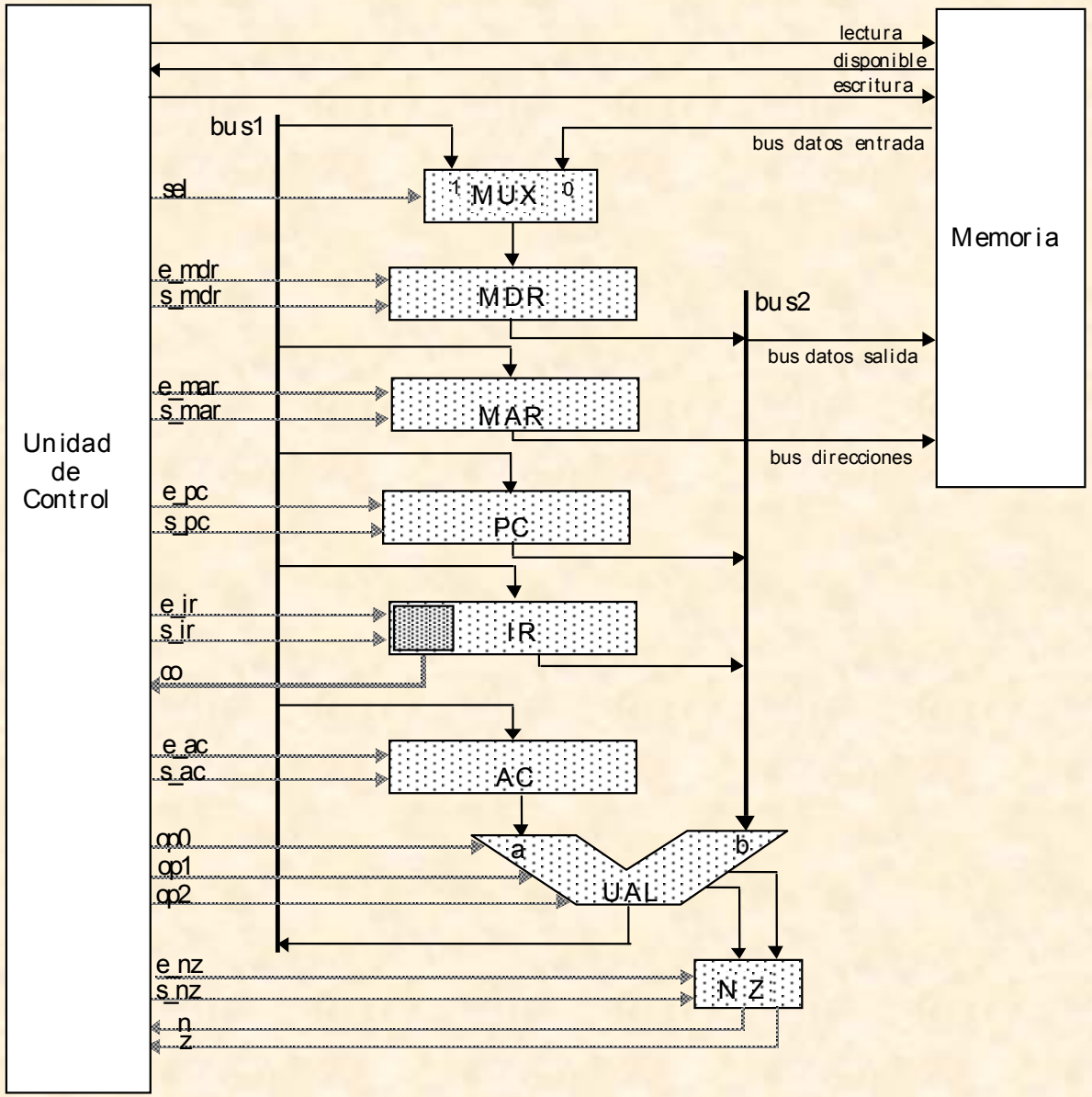
Resultados de la simulación

ns	delta	start	sal	ent	dir	lec	dis	esc	stop	clk
0	+0	0	000000000000	000000000000	000000000000	0	0	0	0	0
5	+0	1	000000000000	000000000000	000000000000	0	0	0	0	0
5	+2	1	000000000000	000000000000	000000000000	0	0	0	0	1
10	+1	1	000000000000	000000000000	000000000000	0	0	0	0	0
20	+1	1	000000000000	000000000000	000000000000	0	0	0	0	1
LDAI 3										
25	+1	1	000000000000	000000000000	000000000000	0	0	0	0	0
35	+1	1	000000000000	000000000000	000000000000	0	0	0	0	1
40	+1	1	000000000000	000000000000	000000000000	0	0	0	0	0
40	+2	1	000000000000	000000000000	000000000000	1	0	0	0	0
50	+1	1	000000000000	000000000000	000000000000	1	0	0	0	1
55	+0	1	000000000000	010000000011	000000000000	1	0	0	0	1
55	+1	1	000000000000	010000000011	000000000000	1	0	0	0	0
60	+0	1	000000000000	010000000011	000000000000	1	1	0	0	0
65	+1	1	000000000000	010000000011	000000000000	1	1	0	0	1
70	+1	1	000000000000	010000000011	000000000000	1	1	0	0	0
80	+1	1	000000000000	010000000011	000000000000	1	1	0	0	1
85	+1	1	000000000000	010000000011	000000000000	1	1	0	0	0
85	+2	1	000000000000	010000000011	000000000000	0	1	0	0	0
90	+0	1	000000000000	000000000000	000000000000	0	0	0	0	0
95	+1	1	000000000000	000000000000	000000000000	0	0	0	0	1
100	+1	1	000000000000	000000000000	000000000000	0	0	0	0	0
110	+1	1	000000000000	000000000000	000000000000	0	0	0	0	1
115	+1	1	000000000000	000000000000	000000000000	0	0	0	0	0
125	+1	1	000000000000	000000000000	000000000000	0	0	0	0	1
130	+1	1	000000000000	000000000000	000000000000	0	0	0	0	0
140	+1	1	000000000000	000000000000	000000000000	0	0	0	0	1
145	+1	1	000000000000	000000000000	000000000000	0	0	0	0	0
SUM 4										
145	+2	1	000000000000	000000000000	000000000001	0	0	0	0	0
155	+1	1	000000000000	000000000000	000000000001	0	0	0	0	1
160	+1	1	000000000000	000000000000	000000000001	0	0	0	0	0
160	+2	1	000000000000	000000000000	000000000001	1	0	0	0	0
170	+1	1	000000000000	000000000000	000000000001	1	0	0	0	1
175	+0	1	000000000000	001100000100	000000000001	1	0	0	0	1

VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (1)

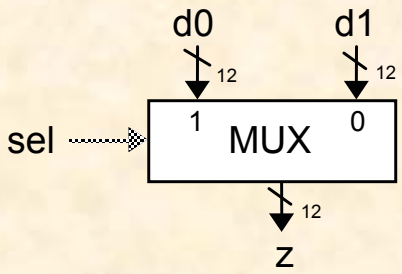
En este modelo vamos a incorporar la estructura interna de la ruta de datos, es decir, sustituiremos el modelo de comportamiento algoritmico de esta unidad por otro estructural compuesto por un conjunto de registros y unidades funcionales interconectados por una red de buses y/o multiplexores



VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (2)

Multiplexor

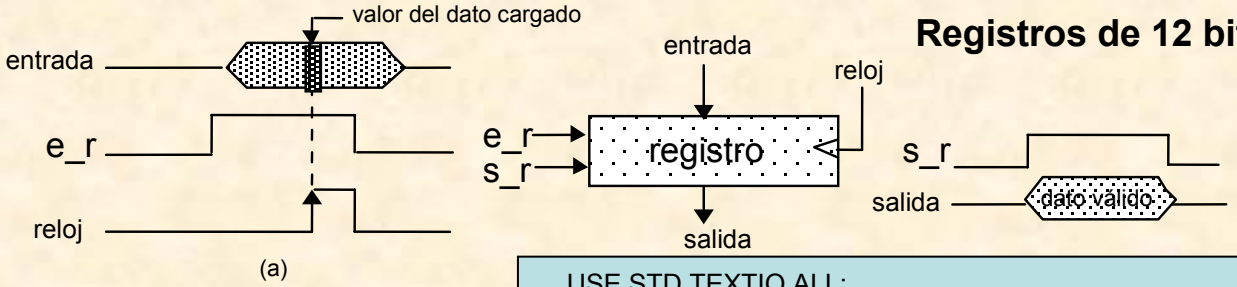


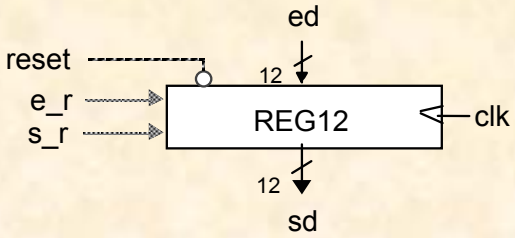

```
USE WORK.std_logic_1164.ALL;
ENTITY mux IS
   PORT (d0, d1 : IN std_logic_vector(11 DOWNTO 0);
         z : OUT std_logic_vector(11 DOWNTO 0);
         sel : IN std_ulogic);
END mux;

ARCHITECTURE comportamiento OF mux IS
BEGIN
   PROCESS(d0, d1, sel)
   BEGIN
      IF sel = '1'
         THEN
            z <= d1 AFTER 5 ns;
         ELSE
            z <= d0 AFTER 5 ns;
      END IF;
   END PROCESS;
END comportamiento;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (3)

Registros de 12 bits (mdr, mar, pc, ac)

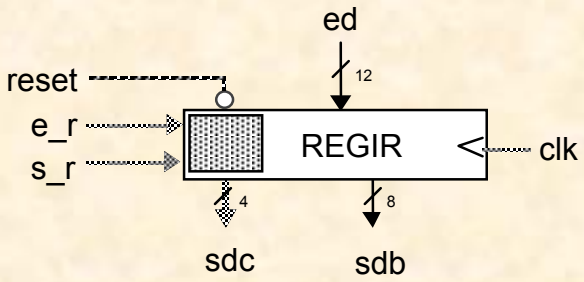





```
USE STD.TEXTIO.ALL;
USE WORK.std_logic_1164.ALL;
ENTITY reg12 IS
   GENERIC ( nombre : STRING := "registro general");
   PORT (  ed : IN std_logic_vector(11 DOWNTO 0);
         clk, reset, e_r, s_r : IN std_ulogic;
         sd : OUT std_logic_vector(11 DOWNTO 0));
END reg12;
ARCHITECTURE comportamiento OF reg12 IS
BEGIN
   PROCESS
      VARIABLE r, ar : std_logic_vector(11 DOWNTO 0) := "000000000000";
      VARIABLE l : LINE;
   BEGIN
      WAIT UNTIL (rising_edge(clk) OR s_r'EVENT OR reset'EVENT);
      ar := r;
      IF falling_edge(reset) THEN r := "000000000000"; END IF;
      IF rising_edge(clk) THEN
         IF e_r = '1' THEN r := ed; END IF;
      END IF;
      IF s_r = '1' THEN sd <= r; ELSE sd <= "ZZZZZZZZZZZZ"; END IF;
      --
      IF r /= ar THEN                                  -- Código
        WRITE(l, nombre);                              -- añadido
        WRITE(l,To_bitvector(r)); -- para
        WRITELINE(OUTPUT,l); -- depuración
      END IF;                                          -- de la RD --
   END PROCESS;
END comportamiento;
```

Modelo Estructural de la Ruta de Datos (4)

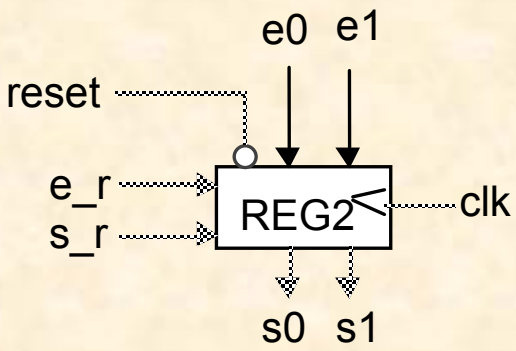
Registro de Instrucciones




```
USE STD.TEXTIO.ALL;
USE WORK.std_logic_1164.ALL;
ENTITY regir IS
   GENERIC (nombre : STRING := "registro instrucción");
   PORT (  ed : IN std_logic_vector(11 DOWNTO 0);
         clk, reset, e_r, s_r : IN std_ulogic;
         sdb : OUT std_logic_vector(7 DOWNTO 0);
         sdc : OUT std_logic_vector(3 DOWNTO 0));
END regir;
ARCHITECTURE comportamiento OF regir IS
BEGIN
   PROCESS
      VARIABLE r, ar : std_logic_vector(11 DOWNTO 0) := "000000000000";
      VARIABLE l : LINE;
   BEGIN
      WAIT UNTIL (rising_edge(clk) OR s_r'EVENT OR reset'EVENT);
      ar := r;
      IF falling_edge(reset) THEN r := "000000000000"; END IF;
      IF rising_edge(clk) THEN
         IF e_r = '1' THEN r := ed; END IF;
      END IF;
      IF s_r = '1'
         THEN
            sdb <= r(7 DOWNTO 0);
         ELSE
            sdb <= "ZZZZZZZZ";
      END IF;
      sdc <= r(11 DOWNTO 8);
      IF r /= ar THEN
        WRITE(l, nombre);
        WRITE(l,To_bitvector(r));
        WRITELINE(OUTPUT,l);
      END IF;
   END PROCESS;
END comportamiento;
```

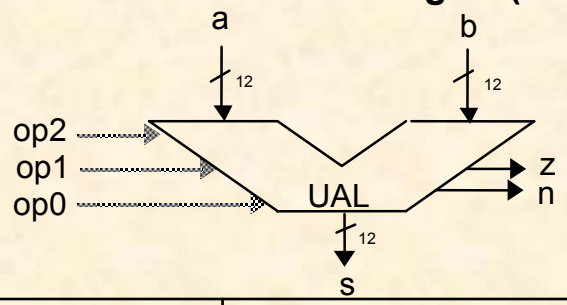
Modelo Estructural de la Ruta de Datos (5)

Registro de Estado




```
USE STD.TEXTIO.ALL;
USE WORK.std_logic_1164.ALL;
ENTITY reg2 IS
   PORT (  e0, e1, clk, reset, e_r, s_r : IN std_ulogic;
                        s0, s1 : OUT std_ulogic);
END reg2;
ARCHITECTURE comportamiento OF reg2 IS
BEGIN
   PROCESS
      VARIABLE r0, r1, ar0, ar1 : std_ulogic := '0';
      VARIABLE l : LINE;
      CONSTANT creg2 : STRING := "<NZ>  = ";
   BEGIN
      WAIT UNTIL (rising_edge(clk) OR s_r'EVENT OR reset'EVENT);
      ar0 := r0; ar1 := r1;
      IF falling_edge(reset) THEN r0 := '0'; r1 := '0'; END IF;
      IF rising_edge(clk) THEN
         IF e_r = '1' THEN r0 := e0; r1 := e1; END IF;
      END IF;
      IF s_r = '1'
         THEN
            s0 <= r0 AFTER 10 ns;
            s1 <= r1 AFTER 10 ns;
         ELSE
            s0 <= 'Z' AFTER 10 ns;
            s1 <= 'Z' AFTER 10 ns;
      END IF;
      IF (r0 /= ar0) OR (r1 /= ar1) THEN
        WRITE(l, creg2);
        WRITE(l,To_bit(r1, '0'));WRITE(l,To_bit(r0,'0'));
        WRITELINE(OUTPUT,l);
      END IF;
   END PROCESS;
END comportamiento;
```

Unidad Aritmético-Lógica (ual)



op2	op1	op0	s
0	0	0	"000000000000"
0	0	1	"0000" & b(7 DOWNTO 0)
0	1	0	a NOR b
0	1	1	b
1	0	0	b + 1
1	0	1	a
1	1	0	a + b
1	1	1	no opera

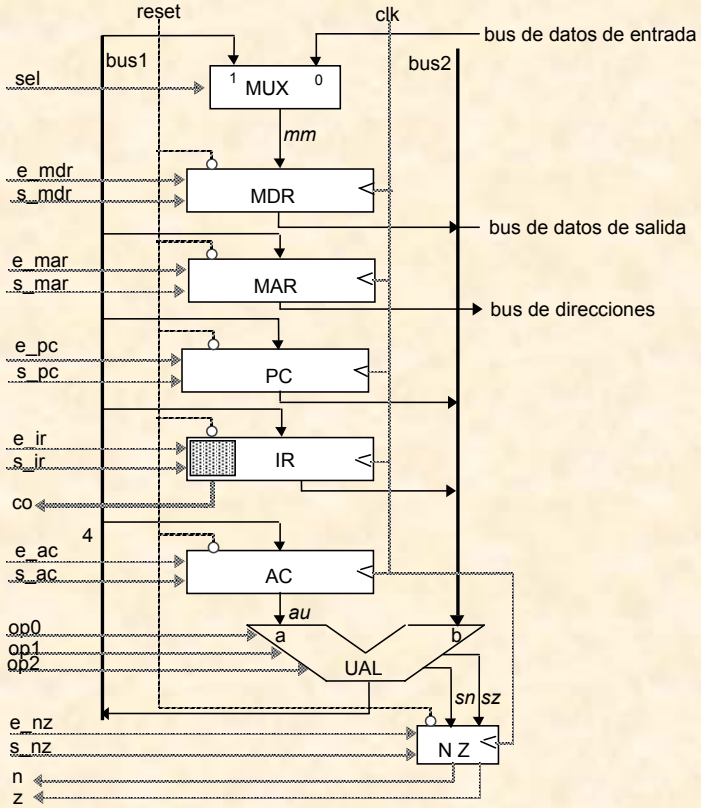
Para compatibilizar tipos, se han utilizado las siguientes funciones de conversión de tipos del paquete STD_LOGC_1164:

- To_bit (,) :
  std_ulogic a bit
- To_StdLogicVector ( ) :
  bit_vector → std_logic_vector
- To_StdULogicVector ( ) :
  std_logic_vector → std_ulogic_vector
- To_bitvector( ) :
  std_logic_vector → bit_vector

```
USE WORK.utilidad.ALL;
USE WORK.std_logic_1164.ALL;
ENTITY ual IS
   PORT(   a, b : IN std_logic_vector(11 DOWNTO 0);
           op0, op1, op2 : IN std_ulogic;
           s : OUT std_logic_vector(11 DOWNTO 0);
           n, z : OUT std_ulogic);
END ual;
ARCHITECTURE comportamiento OF ual IS
BEGIN
   PROCESS(a,b,op0, op1,op2)
   VARIABLE as : std_logic_vector(11 DOWNTO 0);
   VARIABLE op :bit_vector(2 DOWNTO 0);
   BEGIN
      op := To_bit(op2, '0')&To_bit(op1, '0')&To_bit(op0,'0');
      CASE op IS
         WHEN "000" => as := "000000000000";
         WHEN "001" => as := "0000" & b(7 DOWNTO 0);
         WHEN "010" =>
  as := To_StdLogicVector(To_StdULogicVector(a) NOR
To_StdULogicVector(b)) ;
         WHEN "011" => as := b;
         WHEN "100" =>
         as := To_StdLogicVector(n_a_b((b_a_n(To_bitvector(b)) + 1), 12));
         WHEN "101" => as := a;
         WHEN "110" =>
as := To_StdLogicVector(n_a_b(( b_a_n(To_bitvector(a)) +
b_a_n(To_bitvector(b))), 12));
         WHEN OTHERS => NULL;
      END CASE;
      s <= as AFTER 5 ns;
      IF as(11)= '1'  THEN n <= '1' AFTER 5 ns; ELSE n <= '0' AFTER 5 ns;
END IF;
      IF as = "000000000000“ THEN z <= '1' AFTER 5 ns; ELSE z <= '0'
AFTER 5 ns;
      END IF;
   END PROCESS;
END comportamiento;
```

Modelo Estructural de la Ruta de Datos (7)

Conexión estructural de la ruta de datos




```
USE WORK.std_logic_1164.ALL;
USE WORK.ALL;
ENTITY ruta IS
   PORT (  e_dat : OUT std_logic_vector(11 DOWNTO 0);
           s_dat : IN std_logic_vector(11 DOWNTO 0);
           dir : OUT std_logic_vector(11 DOWNTO 0);
           clk, reset, sel, e_mdr, s_mdr, e_mar, s_mar, e_pc, s_pc,
           e_ir, s_ir, e_ac, s_ac, e_nz, s_nz : IN std_ulogic;
           op0, op1, op2 : IN std_ulogic;
           n, z : OUT std_logic;
           co : OUT std_logic_vector(3 DOWNTO 0));
END ruta;
```

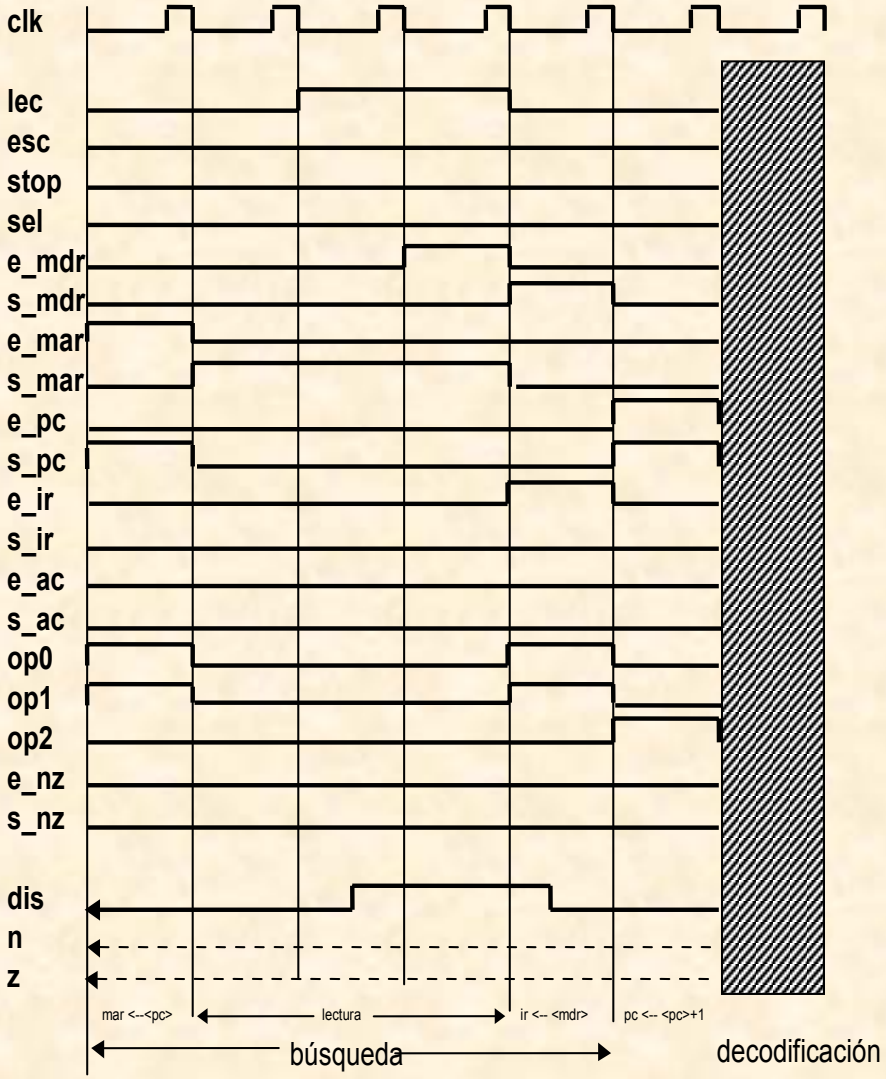
```
ARCHITECTURE estructura OF ruta IS
   COMPONENT mux
      PORT (  d0, d1 : IN std_logic_vector(11 DOWNTO 0);
              z : OUT std_logic_vector(11 DOWNTO 0);
              sel : IN std_ulogic);
   END COMPONENT;
   COMPONENT reg12
      GENERIC (nombre : STRING);
      PORT (  ed : IN std_logic_vector(11 DOWNTO 0);
              clk, reset, e_r, s_r : IN std_ulogic;
              sd : OUT std_logic_vector(11 DOWNTO 0) );
   END COMPONENT;
   COMPONENT regir
      GENERIC ( nombre : STRING);
      PORT (  ed : IN std_logic_vector(11 DOWNTO 0);
              clk, reset, e_r, s_r : IN std_ulogic;
              sdb : OUT std_logic_vector(7 DOWNTO 0);
              sdc : OUT std_logic_vector(11 DOWNTO 8));
   END COMPONENT;
   COMPONENT reg2
      PORT (  e0, e1 : IN std_logic;
              clk, reset, e_r, s_r : IN std_ulogic;
              s0, s1 : OUT std_logic );
   END COMPONENT;
   COMPONENT ual
      PORT(   a, b : IN std_logic_vector(11 DOWNTO 0);
              op0, op1, op2 : IN std_ulogic;
              s : OUT std_logic_vector(11 DOWNTO 0);
              n, z : std_logic);
   END COMPONENT;
   SIGNAL bus1, bus2, mm, au : std_logic_vector(11 DOWNTO 0);
   SIGNAL sn, sz : std_logic;
   CONSTANT rmdr : STRING := "<MDR> = ";
   CONSTANT rmar : STRING := "<MAR> = ";
   CONSTANT rpc  : STRING := "<PC>  = ";
   CONSTANT rac  : STRING := "<AC>  = ";
   CONSTANT rir  : STRING := "<IR>  = ";
BEGIN
   mu  : mux PORT MAP(s_dat, bus1, mm, sel);
   mdr : reg12 GENERIC MAP(rmdr) PORT MAP(mm, clk, reset, e_mdr, s_mdr, bus2);
   mar : reg12 GENERIC MAP(rmar) PORT MAP(bus1, clk, reset, e_mar, s_mar, dir);
   pc  : reg12 GENERIC MAP(rpc) PORT MAP(bus1, clk, reset, e_pc, s_pc, bus2);
   ir  : regir GENERIC MAP (rir)
   PORT MAP(bus1, clk, reset, e_ir, s_ir, bus2(7 DOWNTO 0), co);
   ac  : reg12 GENERIC MAP(rac) PORT MAP(bus1, clk, reset, e_ac, s_ac, au);
   ua  : ual PORT MAP(au,bus2, op0, op1, op2, bus1, sn, sz);
   nz  : reg2 PORT MAP(sz, sn, clk, reset, e_nz, s_nz, n, z);
   e_dat <= bus2;
END estructura;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (8)
Modelo de la Unidad de Control

• La Unidad de Control se encarga de generar todas las señales que gobiernan el flujo de información por la *RD*.

Fases de búsqueda y decodificación: se lee la instrucción de memoria, se transfiere a *IR y se decodifica*.



<PC> → MAR

- *s_pc* (salir del *PC*)
- *op0* y *op1* (paso transparente por *b* de la *UAL*)
- *e_mar* (entrar en el *MAR*).

Lectura de la instrucción

- *s_mar* (salir del *MAR)*
- *lec* (al comienzo del siguiente ciclo)
- *dis*, (disponibilidad del dato)
- *e_mdr* (cargar el dato leido en el registro *MDR*.)
- desactivación de *s_mar* y *lec*

<MDR> → *RI*

- *s_mdr*, *op0*, *op1* y *e_ri*.

<PC> + 1 → PC

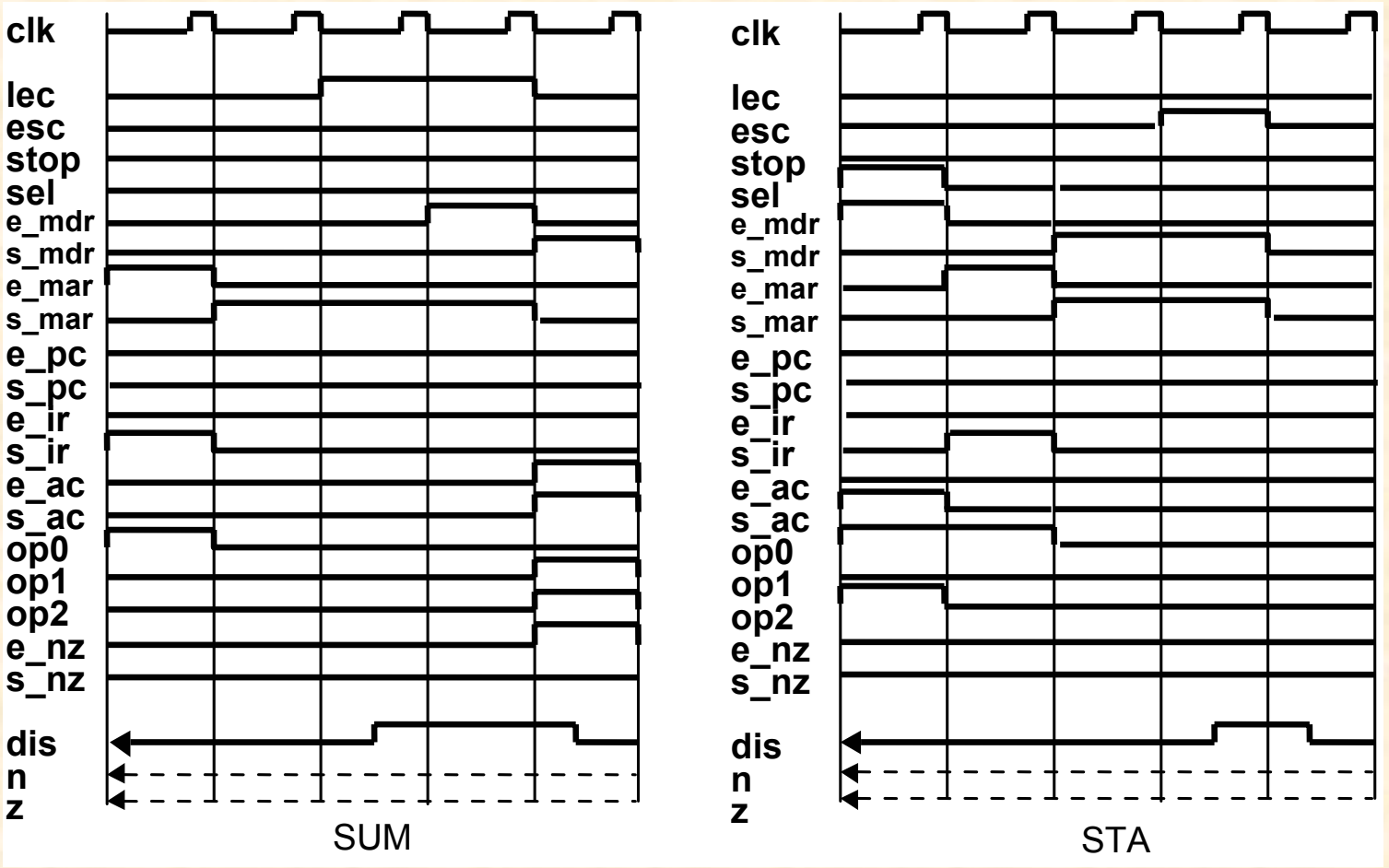
- *s_pc*, *op2* y *e_pc*.

Decodificación

- la *UC* generará las señales específicas que gobiernan el flujo de información para la correspondiente instrucción

Modelo Estructural de la Ruta de Datos (9)

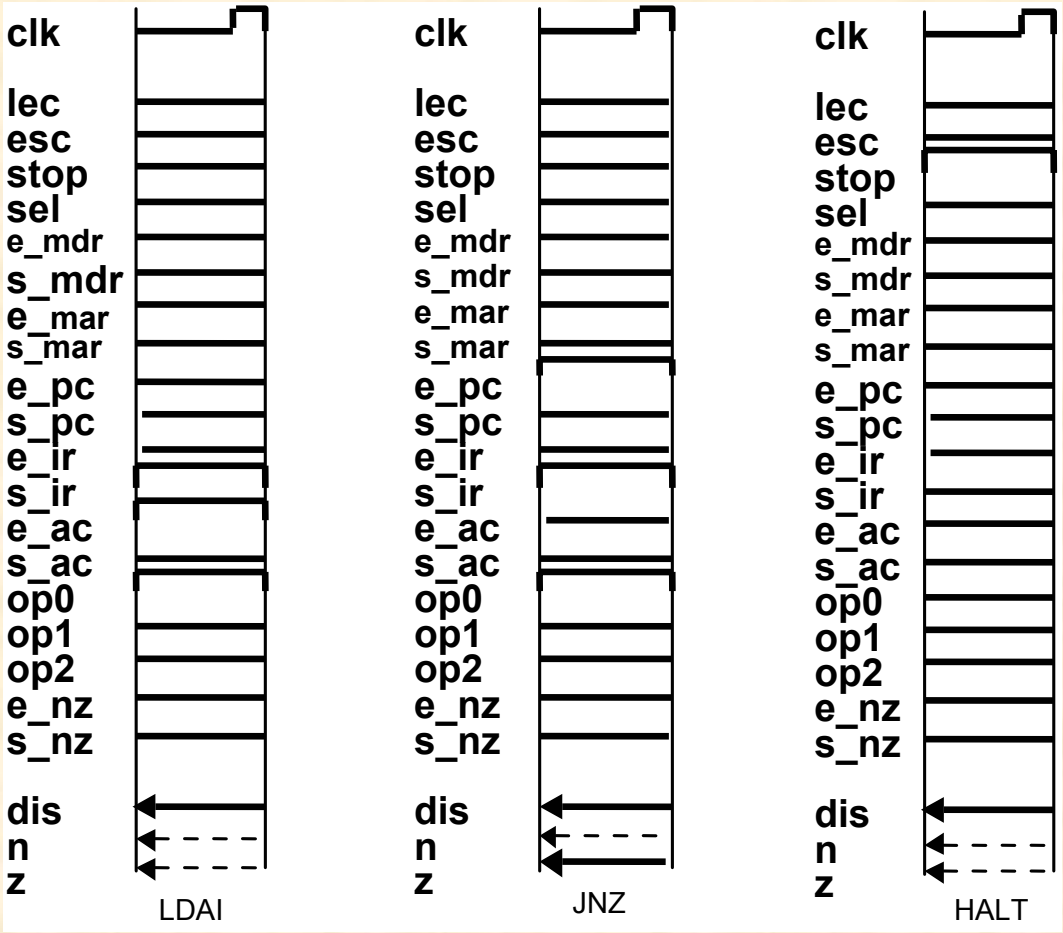
Cronogramas correspondientes a las instrucciones *SUM* y *STA*



VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (10)

Cronogramas correspondientes a las instrucciones *LDAI*, *JNZ* y *HALT*.



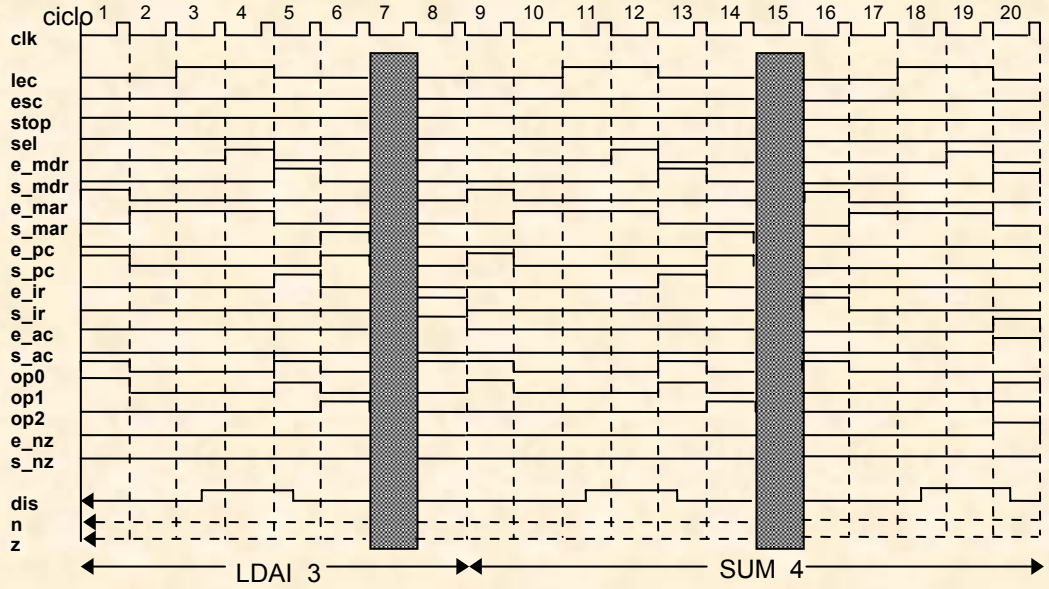
VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

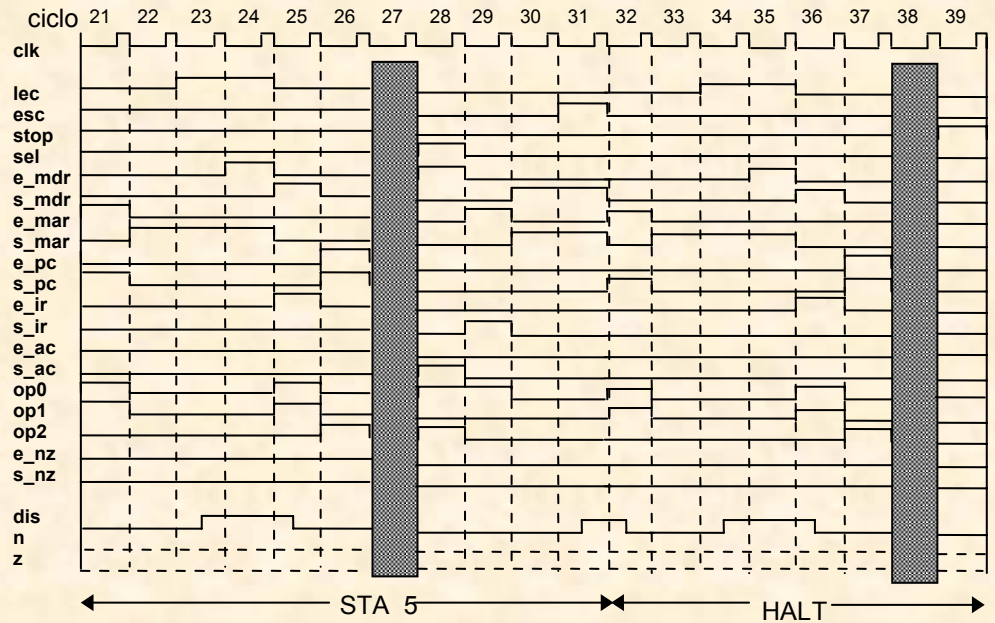
Modelo Estructural de la Ruta de Datos (11)

Señales generadas por la *UC* a lo largo de la ejecución delsiguiente programa:

Dir.	*Simbólico*	*Binario*
0	LDAI 3	0100 00000011
1	SUM 4	0011 00000100
2	STA 5	0010 00000101
3	HALT	0000 00000000

- El programa dura 39 ciclos
- La parada la origina la ejecución de la instrucción *HALT* activando la señal *stop*, que bloquea la generación de pulsos de reloj





VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (12)

Partedel paquete standard IEEE std_logic_1164

```
PACKAGE std_logic_1164 IS
TYPE std_ulogic IS ( 'U',  -- Uninitialized
                     'X',  -- Forcing  Unknown
                     '0',  -- Forcing  0
                     '1',  -- Forcing  1
                     'Z',  -- High Impedance
                     'W',  -- Weak     Unknown
                     'L',  -- Weak     0
                     'H',  -- Weak     1
                     '-'   -- Don't care   );
  TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;

 -- resolution function
  FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;
  SUBTYPE std_logic IS resolved std_ulogic;
  TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_logic;

 -- overloaded logical operators
  FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
  FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
  FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
  FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
  FUNCTION "xor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
  FUNCTION "not"  ( l : std_ulogic                 ) RETURN UX01;

  -- conversion functions
  FUNCTION To_bit       ( s : std_ulogic;       xmap : BIT := '0') RETURN BIT;
  FUNCTION To_bitvector ( s : std_logic_vector ; xmap : BIT := '0') RETURN BIT_VECTOR;
  FUNCTION To_bitvector ( s : std_ulogic_vector; xmap : BIT := '0') RETURN BIT_VECTOR;
  FUNCTION To_StdULogic       ( b : BIT              ) RETURN std_ulogic;
  FUNCTION To_StdLogicVector  ( b : BIT_VECTOR       ) RETURN std_logic_vector;
  FUNCTION To_StdLogicVector  ( s : std_ulogic_vector ) RETURN std_logic_vector;
  FUNCTION To_StdULogicVector ( b : BIT_VECTOR       ) RETURN std_ulogic_vector;
  FUNCTION To_StdULogicVector ( s : std_logic_vector  ) RETURN std_ulogic_vector;

 -- edge detection
  FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN;
  FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN;
  ----------------------------------------------------------------
END std_logic_1164;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (13)

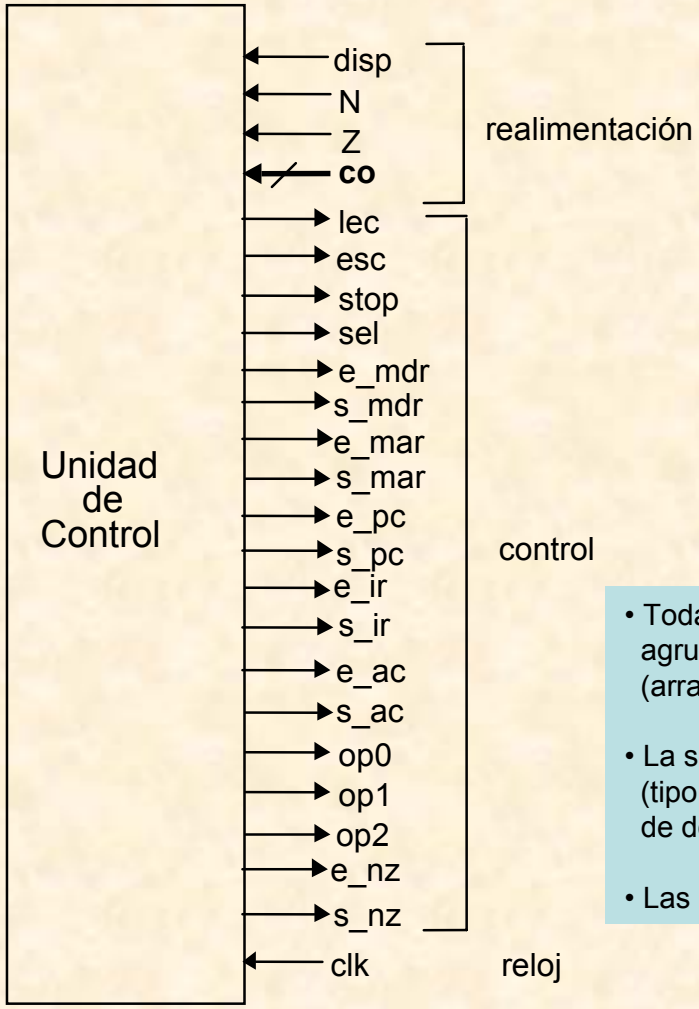
Definición de los tipos de control

```
USE WORK.std_logic_1164.ALL;
PACKAGE tipos_control IS
TYPE senales_control IS (lec, esc, stop, sel, e_mdr, s_mdr, e_mar,
                         s_mar,e_pc, s_pc, e_ir, s_ir, e_ac, s_ac,
                         op0, op1, op2, e_nz, s_nz);
TYPE vector_senales_control IS ARRAY (natural RANGE <>) OF senales_control;
TYPE bus_control IS ARRAY (senales_control) OF std_ulogic;
FUNCTION ctrl (ent : vector_senales_control) RETURN bus_control;
FUNCTION ctrl (ent : senales_control) RETURN bus_control;
FUNCTION ctrl RETURN bus_control;
END tipos_control;
PACKAGE BODY tipos_control IS
FUNCTION ctrl(ent : vector_senales_control) RETURN bus_control IS
   VARIABLE res : bus_control := (OTHERS => '0');
      BEGIN
            FOR i IN ent'RANGE LOOP res(ent(i)) := '1'; END LOOP;
            RETURN res;
      END ctrl;
FUNCTION ctrl(ent : senales_control) RETURN bus_control IS
VARIABLE res : bus_control := (OTHERS => '0');
BEGIN
      res(ent) := '1';
      RETURN res;
END ctrl;
FUNCTION ctrl RETURN bus_control IS
      VARIABLE res : bus_control := (OTHERS => '0');
BEGIN
      RETURN res;
END ctrl;
END tipos_control;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (14)

Modelo de comportamiento de la Unidad de Control



Declaración de entidad

```
USE WORK.std_logic_1164.ALL;
USE WORK.tipos_control.ALL;
ENTITY u_control IS
        PORT(disp, n, z : IN std_ulogic;
             co : IN std_logic_vector(3 DOWNTO 0);
             control : OUT bus_control;
             clk : IN std_ulogic);
END u_control;
```

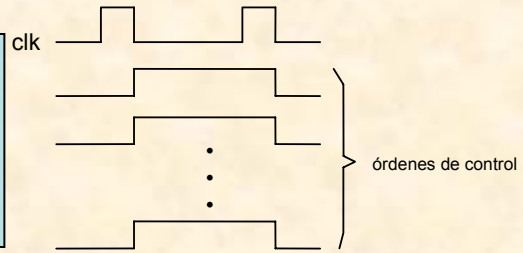
- Todas las señales binarias de salida (tipo *std_ulogic*) las hemos agrupado en un *array* de nombre *control* de tipo *bus_control* (array de *std_ulogic* sobre un índice enumerado de nombres de señales).
- La señal de entrada *co* será la única resuelta de tipo *std_logic_vector* (tipo utilizado en la salida de los registros de la *RD*, en particular el *IR*, de donde procede *co*)
- Las restantes entradas (*disp*, *N*, *Z*, y *clk*) serán todas del tipo *std_ulogic*.

VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

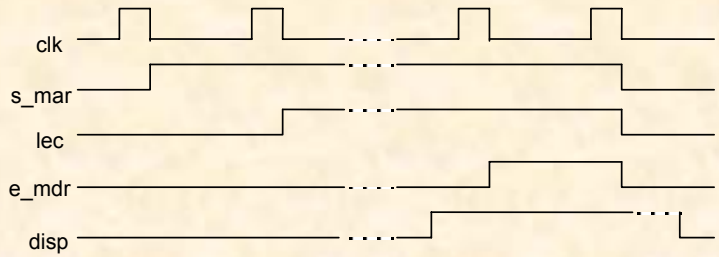
Modelo Estructural de la Ruta de Datos (15)

Arquitectura: procedimientos auxiliares

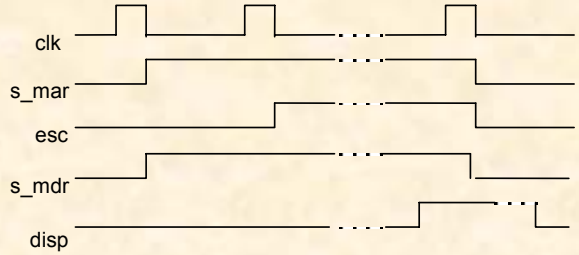
```
PROCEDURE m_orden (ordenes : IN vector_senales_control) IS
   BEGIN
      WAIT UNTIL falling_edge(clk);
      control <= ctrl(ordenes);
   END m_orden;
```




```
PROCEDURE lectura IS
   BEGIN
      WAIT UNTIL falling_edge(clk);
      control <= ctrl(s_mar & s_mar);
      WAIT UNTIL falling_edge(clk);
      control <= ctrl(s_mar & lec);
      WAIT UNTIL disp = '1';
      WAIT UNTIL falling_edge(clk);
      control <= ctrl(s_mar & lec & e_mdr);
   END lectura;
```




```
PROCEDURE escritura IS
   BEGIN
      WAIT UNTIL falling_edge(clk);
      control <= ctrl(s_mar & s_mdr);
      WAIT UNTIL falling_edge(clk);
      control <= ctrl(s_mar & s_mdr & esc);
      WAIT UNTIL disp = '1';
   END escritura
```



VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Arquitectura

```
ARCHITECTURE comportamiento OF u_control IS BEGIN
  PROCESS
   PROCEDURE m_orden(ordenes : IN vector_senales_control) IS
    BEGIN
       WAIT UNTIL falling_edge(clk);
       control <= ctrl(ordenes);
    END m_orden;
       PROCEDURE lectura IS
    BEGIN
       WAIT UNTIL falling_edge(clk);
       control <= ctrl(s_mar & s_mar);
       WAIT UNTIL falling_edge(clk);
       control <= ctrl(s_mar & lec);
       WAIT UNTIL disp = '1';
       WAIT UNTIL falling_edge(clk);
       control <= ctrl(s_mar & lec & e_mdr);
    END lectura;
    PROCEDURE escritura IS
    BEGIN
       WAIT UNTIL falling_edge(clk);
       control <= ctrl(s_mar & s_mdr);
       WAIT UNTIL falling_edge(clk);
       control <= ctrl(s_mar & s_mdr & esc);
       WAIT UNTIL disp = '1';
    END escritura;
BEGIN
```

```
m_orden(s_pc & op0 & op1 & e_mar);                    -- mar <-- <pc>
      lectura;                                        -- Lectura
      m_orden(s_mdr & op0 & op1 & e_ir);              -- ir  <-- <mdr>
      m_orden(s_pc & op2 & e_pc);                     -- pc  <-- <pc> + 1
      WAIT UNTIL falling_edge(clk);                   -- Decodificación
                  control <= ctrl;
      CASE co IS
        WHEN "0001" =>                                -- LDA
          m_orden(s_ir & op0 & e_mar);                -- mar <-- <ir>
          lectura;                                    -- Lectura
          m_orden(s_mdr & op0 & op1 & e_ac);          -- ac <-- <mdr>
        WHEN "0010" =>                                -- STA
          m_orden(s_ac & op0 & op2 & sel & e_mdr);    -- mdr <-- <ac>
          m_orden(s_ir & op0 & e_mar);                -- mar <-- <ir>
          escritura;                                  -- Escritura
        WHEN "0011" =>                                -- SUM
          m_orden(s_ir & op0 & e_mar);                -- mar <-- <ir>
          lectura;                                    -- Lectura
          m_orden(s_mdr & s_ac & op2 & op1 & e_ac & e_nz); -- ac <-- <ac>+<mdr>
        WHEN "0100" =>                                -- LDAI
          m_orden(s_ir & op0 & e_ac);                 -- ac <-- <ir>
        WHEN "0101" =>                                -- SUMI
          m_orden(s_ir & op0 & e_mdr);                -- mdr <-- <ir>
          m_orden(s_mdr & s_ac & op2 & op1 & e_ac & e_nz);-- ac <-- <ac>+<mdr>
        WHEN "0110" =>                                -- NOR
          m_orden(s_ir & op0 & e_mar);                -- mar <-- <ir>
          lectura;                          -- lectura
          m_orden(s_mdr & s_ac & op1 & e_ac & e_nz);  -- ac <-- <mdr>
        WHEN "0111" =>                                -- JZ
          IF z = '1' THEN
            m_orden(s_ir & op0 & e_pc);               -- pc <-- <ir>
          END IF;
        WHEN "1000" =>                                -- JNZ
          IF z = '1' THEN
            m_orden(s_ir & op0 & e_pc);               -- pc <-- <ir>
          END IF;
        WHEN "1001" =>                                -- JN
          IF n = '1' THEN
            m_orden(s_ir & op0 & e_pc);               -- pc <-- <ir>
          END IF;
        WHEN "1010" =>                                -- JNN
          IF n = '1' THEN
            m_orden(s_ir & op0 & e_pc);               -- pc <-- <ir>
          END IF;
        WHEN OTHERS =>
          m_orden(lec & esc);                         -- Volcado memoria
          m_orden(stop & stop);                       -- Parada
      END CASE;
  END PROCESS;
END comportamiento;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (17)

Reloj

El reloj será el mismo que el utilizado en el modelo anterior pero cambiando el tipo *bit* por el tipo *std_ulogic* en la periferia para compatibilizar sus tipos con los de la *UC* a la que habrá que conectarlo

```
USE WORK.std_logic_1164.ALL;
  ENTITY reloj IS
      GENERIC(tl : TIME := 20 ns; th : TIME := 5 ns);
      PORT(start : IN bit; stop : IN std_ulogic := '0'; clock : OUT std_ulogic);
  END reloj;
  ARCHITECTURE comportamiento OF reloj IS
  SIGNAL clk : bit := '0';
  BEGIN
    PROCESS(start, stop, clk)
    VARIABLE clke : bit := '0';
    BEGIN
      IF (start = '1' and NOT start'STABLE) THEN
      clke := '1';
      clk <= TRANSPORT '1' AFTER 0 ns;
      clk <= TRANSPORT '0' AFTER th;
      END IF;
      IF (stop = '1' and NOT stop'STABLE) THEN
      clke := '0';
      clk <= TRANSPORT '0' AFTER 0 ns;
      END IF;
      IF (clk = '0' and NOT clk'STABLE and clke = '1') THEN
      clk <= TRANSPORT '1' AFTER tl;
      clk <= TRANSPORT '0' AFTER tl+th;
      END IF;
      clock <= To_StdUlogic(clk);
    END PROCESS;
  END comportamiento;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (18)

Al igual que hicimos con el reloj, utilizaremos el mismo modelo anterior cambiando el tipo *bit_vector* por *std_logic_vector* en los buses de datos (*bus_dat_sal*, *bus_dat_ent*) y direcciones (*bus_dir*), y el tipo *bit* por el *std_ulogic* en las lineas de control (*lec*, *esc*, *dis*):

```
USE STD.TEXTIO.aLL;
USE WORK.std_logic_1164.ALL;
USE WORK.utilidad.ALL;
ENTITY memoria IS
  PORT (bus_dat_sal : OUT std_logic_vector(11 DOWNTO 0);
        bus_dat_ent :  IN std_logic_vector(11 DOWNTO 0);
        bus_dir : IN std_logic_vector(11 DOWNTO 0);
        lec, esc : IN std_ulogic;
        dis : OUT std_ulogic);
END memoria;
ARCHITECTURE comportamiento OF memoria IS
BEGIN
  PROCESS
    TYPE array_memoria IS
    ARRAY (NATURAL RANGE 0 TO 255) OF
                std_logic_vector(11 DOWNTO 0);
    VARIABLE mem : array_memoria :=
                    -- programa a ejecutar
                        ("000100000100", --  0 LDA (STA índice)
                         "010100000001", --  1 SUMI 1
                         "001000000100", --  2 STA (STA índice+1)
                         "000100001110", --  3 LDA índice
                         "001000010011", --  4 STA índice
                         "010100000001", --  5 SUMI 1
                         "001000001110", --  6 STA índice
                         "000100001101", --  7 LDA limite
                         "011000001101", --  8 NOR limite
                         "010100000001", --  9 SUMI 1
                         "001100001110", -- 10 SUM índice
                         "100000000000", -- 11 JNZ 0
                         "000000000000", -- 12 HALT
                         "000000011110", -- 13 limite
                         "000000010100", -- 14 índice
                         OTHERS => "000000000000");
```

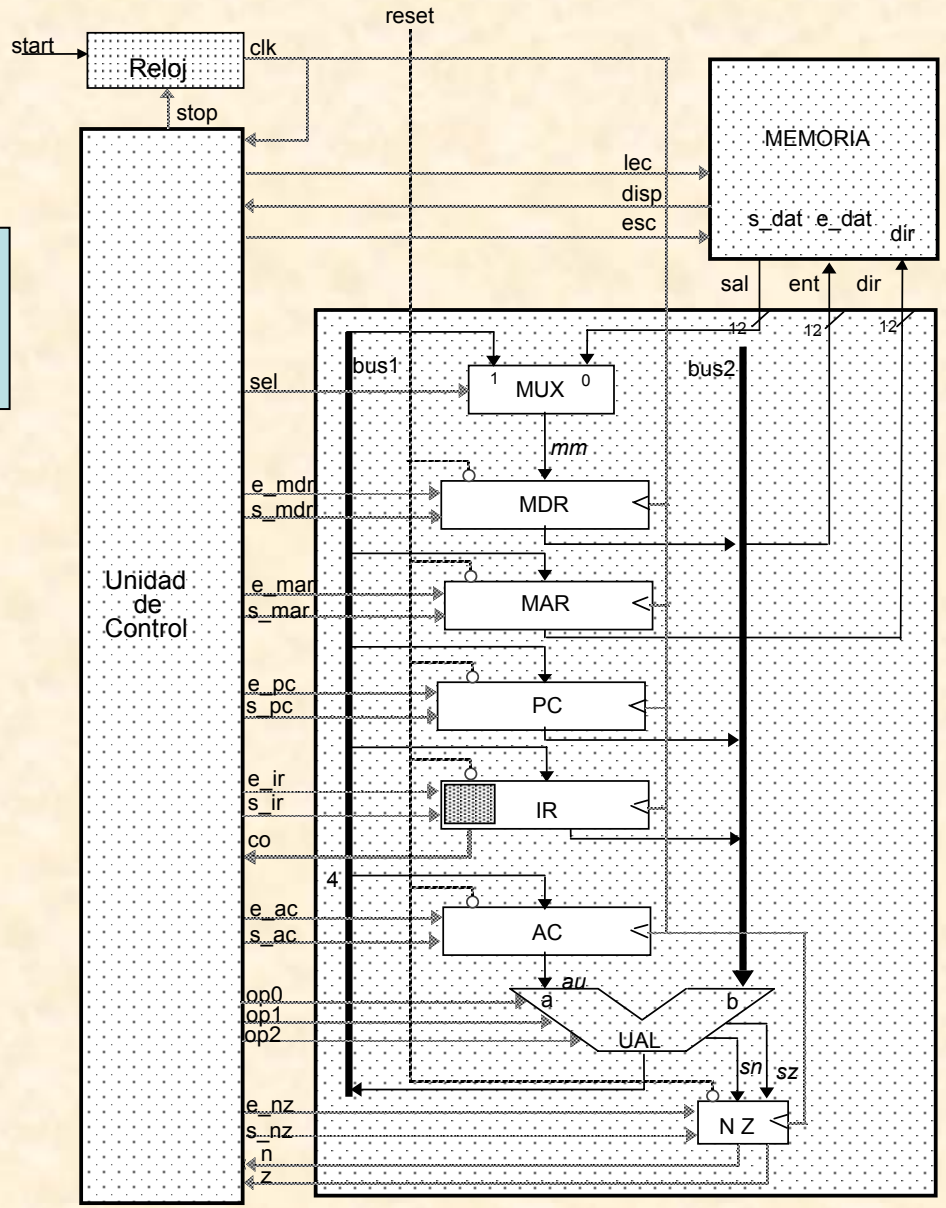
```
SUBTYPE dir IS NATURAL RANGE 0 TO 255;
   VARIABLE direc : dir;
   VARIABLE l : LINE;
   CONSTANT c : STRING := " = ";
  BEGIN
   bus_dat_sal <= "ZZZZZZZZZZZZ" AFTER 5 ns;
   dis <= '0'AFTER 5 ns;
   WAIT UNTIL (lec = '1') OR (esc = '1');
   direc := b_a_n(To_bitvector(bus_dir,'0'));
      -- Visualiza las 20 primeras posiciones de memoria
      IF (esc = '1') AND (lec = '1')  THEN            --
         FOR i IN 0 TO 19 LOOP                        --
            WRITE(l,i);                               --
            WRITE(l, c);                              --
            WRITE(l, To_bitvector(mem(i)));           --
            WRITELINE(OUTPUT, l);                     --
         END LOOP;                                    --
      -- cuando se activan simultáneamente  lec  y  esc
      ELSIF esc = '1' THEN
         mem(direc) := bus_dat_ent;
         dis <= '1' AFTER 20 ns;
         WAIT UNTIL esc = '0';
      ELSE
         bus_dat_sal <= mem(direc) AFTER 15 ns;
         dis <= '1' AFTER 20 ns;
         WAIT UNTIL lec = '0';
      END IF;
  END PROCESS;
END comportamiento;
```

Modelo Estructural de la Ruta de Datos (19)

Conexión estructural Reloj-Memoria-Unidad de Control-Ruta de Datos

Declaración de entidad

```
USE WORK.std_logic_1164.ALL;
USE WORK.tipos_control.ALL;
ENTITY comput IS
      PORT(start : IN bit; reset : IN std_ulogic := '1');
END comput;
```



Modelo Estructural de la Ruta de Datos (20)

```
ARCHITECTURE estructural OF comput IS
  COMPONENT reloj
      GENERIC(tl,th : TIME := 10 ns);
      PORT(start : IN bit; stop : IN std_ulogic; clock : OUT
std_ulogic);
  END COMPONENT;
   COMPONENT u_control
      PORT(disp, n, z : IN std_logic;
          co : IN std_logic_vector(3 DOWNTO 0);
          control : OUT bus_control;
          clk : IN std_ulogic);
   END COMPONENT;
   COMPONENT ruta
      PORT (  e_dat : OUT std_logic_vector(11 DOWNTO 0);
            s_dat : IN std_logic_vector(11 DOWNTO 0);
            dir : OUT std_logic_vector(11 DOWNTO 0);
            clk, reset, sel, e_mdr, s_mdr, e_mar, s_mar, e_pc,
            s_pc, e_ir, s_ir, e_ac, s_ac,
            op0, op1, op2, e_nz, s_nz : IN std_ulogic;
            n, z : OUT std_logic;
            co : OUT std_logic_vector(3 DOWNTO 0));
   END COMPONENT;
   COMPONENT memoria
      PORT (bus_dat_sal : OUT std_logic_vector(11 DOWNTO 0);
          bus_dat_ent :  IN std_logic_vector(11 DOWNTO 0);
          bus_dir : IN std_logic_vector(11 DOWNTO 0);
          lec, esc : IN std_ulogic;
          dis : OUT std_ulogic);
   END COMPONENT;
```

Arquitectura

```
SIGNAL control : bus_control;
   SIGNAL sal, ent : std_logic_vector(11 DOWNTO 0);
   SIGNAL dir : std_logic_vector(11 DOWNTO 0);
   SIGNAL dis, clk : std_ulogic;
   SIGNAL n, z : std_logic;
   SIGNAL co : std_logic_vector(3 DOWNTO 0);
BEGIN
   rel : reloj GENERIC MAP(10 ns, 5 ns) PORT MAP(start, control(stop), clk);
   mem : memoria PORT MAP(sal, ent, dir, control(lec), control(esc), dis);
   con : u_control PORT MAP(dis, n, z, co, control, clk);
   dat : ruta PORT MAP(ent, sal, dir, clk, reset, control(sel),
                  control(e_mdr), control(s_mdr), control(e_mar),
                  control(s_mar), control(e_pc), control(s_pc),
                  control(e_ir), control(s_ir), control(e_ac),
                  control(s_ac), control(op0), control(op1),
                  control(op2), control(e_nz), control(s_nz),
                  n, z, co);
END estructural;
```


VHDL: de la tecnología a la arquitectura de computadores. José J. Ruz Ortiz, Síntesis 1997

Modelo Estructural de la Ruta de Datos (21)

Evolución de buses y señales de control del modelo estructural en la ejecución del programa

ns	delta	start	reset	control (lec esc stop sel e_mdr s_mdr e_mar s_mar e_pc s_pc e_ir s_ir e_ac s_ac op0 op1 op2 e_nz s_nz)	sal	ent	dir	dis	clk	n	z	co
0	+0	0	1	UUUUUUUUUUUUUUUUUUU	UUUUUUUUUUUU	UUUUUUUUUUUU	UUUUUUUUUUUU	U	U	U	U	UUUU
0	+1	0	1	UUUUUUUUUUUUUUUUUUU	UUUUUUUUUUUU	UUUUUUUUUUUU	UUUUUUUUUUUU	U	0	U	U	UUUU
5	+0	1	1	UUUUUUUUUUUUUUUUUUU	ZZZZZZZZZZZZ	UUUUUUUUUUUU	UUUUUUUUUUUU	0	0	U	U	UUUU
5	+2	1	1	UUUUUUUUUUUUUUUUUUU	ZZZZZZZZZZZZ	UUUUUUUUUUUU	UUUUUUUUUUUU	0	1	U	U	UUUU
5	+3	1	1	UUUUUUUUUUUUUUUUUUU	ZZZZZZZZZZZZ	UUUUUUUUUUUU	ZZZZZZZZZZZZ	0	1	U	U	0000
5	+4	1	1	UUUUUUUUUUUUUUUUUUU	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	0	1	U	U	0000
10	+1	1	1	UUUUUUUUUUUUUUUUUUU	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	0	0	U	U	0000
10	+2	1	1	00000001001000011000	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	0	0	U	U	0000
10	+4	1	1	00000001001000011000	ZZZZZZZZZZZZ	000000000000	ZZZZZZZZZZZZ	0	0	U	U	0000
15	+0	1	1	00000001001000011000	ZZZZZZZZZZZZ	000000000000	ZZZZZZZZZZZZ	0	0	Z	Z	0000
20	+1	1	1	00000001001000011000	ZZZZZZZZZZZZ	000000000000	ZZZZZZZZZZZZ	0	1	Z	Z	0000
25	+1	1	1	00000001001000011000	ZZZZZZZZZZZZ	000000000000	ZZZZZZZZZZZZ	0	0	Z	Z	0000
25	+2	1	1	00000001000000000000	ZZZZZZZZZZZZ	000000000000	ZZZZZZZZZZZZ	0	0	Z	Z	0000
25	+3	1	1	00000001000000000000	ZZZZZZZZZZZZ	000000000000	000000000000	0	0	Z	Z	0000
25	+4	1	1	00000001000000000000	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	000000000000	0	0	Z	Z	0000
35	+1	1	1	00000001000000000000	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	000000000000	0	1	Z	Z	0000
40	+1	1	1	00000001000000000000	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	000000000000	0	0	Z	Z	0000
40	+2	1	1	10000001000000000000	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	000000000000	0	0	Z	Z	0000
48	+0	1	1	10000001000000000000	010000000011	ZZZZZZZZZZZZ	000000000000	0	0	Z	Z	0000
50	+0	1	1	10000001000000000000	010000000011	ZZZZZZZZZZZZ	000000000000	1	0	Z	Z	0000
50	+1	1	1	10000001000000000000	010000000011	ZZZZZZZZZZZZ	000000000000	1	1	Z	Z	0000
55	+1	1	1	10000001000000000000	010000000011	ZZZZZZZZZZZZ	000000000000	1	0	Z	Z	0000
55	+2	1	1	10001001000000000000	010000000011	ZZZZZZZZZZZZ	000000000000	1	0	Z	Z	0000
65	+1	1	1	10001001000000000000	010000000011	ZZZZZZZZZZZZ	000000000000	1	1	Z	Z	0000
70	+1	1	1	10001001000000000000	010000000011	ZZZZZZZZZZZZ	000000000000	1	0	Z	Z	0000
70	+2	1	1	00000100001000110000	010000000011	ZZZZZZZZZZZZ	000000000000	1	0	Z	Z	0000
70	+3	1	1	00000100001000110000	010000000011	ZZZZZZZZZZZZ	ZZZZZZZZZZZZ	1	0	Z	Z	0000
70	+4	1	1	00000100001000110000	010000000011	010000000011	ZZZZZZZZZZZZ	1	0	Z	Z	0000
75	+0	1	1	00000100001000110000	ZZZZZZZZZZZZ	010000000011	ZZZZZZZZZZZZ	0	0	Z	Z	0000
80	+1	1	1	00000100001000110000	ZZZZZZZZZZZZ	010000000011	ZZZZZZZZZZZZ	0	1	Z	Z	0000
80	+2	1	1	00000100001000110000	ZZZZZZZZZZZZ	010000000011	ZZZZZZZZZZZZ	0	1	Z	Z	0100
85	+1	1	1	00000100001000110000	ZZZZZZZZZZZZ	010000000011	ZZZZZZZZZZZZ	0	0	Z	Z	0100
85	+2	1	1	00000000110000000100	ZZZZZZZZZZZZ	010000000011	ZZZZZZZZZZZZ	0	0	Z	Z	0100
85	+4	1	1	00000000110000000100	ZZZZZZZZZZZZ	000000000000	ZZZZZZZZZZZZ	0	0	Z	Z	0100

Practica 8 : Diseño de un Computador con VHDL

Objetivos:

Diseño de un computador.

Práctica a realizar:

Diseñar un computador RR (carga-almacenamiento) a tres niveles de resolución:

1. Comportamiento
2. Memoria (comportamiento) -CPU (comportamiento) -Reloj (comportamiento)
3. Memoria (comportamiento) -CPU (estructural) -Reloj (comportamiento)

La especificación de la arquitectura (instrucciones, direccionamientos, etc.) será libre. Como sugerencia se puede utilizar un subconjunto del DLX como el que se presenta en la transparencia siguiente

Resultados a entregar:

Documentación del diseño incluyendo:

1. Especificación completa y precisa de la arquitectura del computador.
2. Listado VHDL comentado de los programas correspondientes a cada modelo.
3. Visualización de los resultados y/o valores de los buses y registros en el proceso de ejecución de un programa de test.

Arquitectura del DLX (subconjunto)

Repertorio de Instrucciones

Instrucción	Simbólico	Operación
Menor que (R)	slt r1, r2, r3	if (r2 < r3) r1<-1 else r1<-0
Menor que (I)	slti r1, r2, #n	if (r2 < n) r1<-1 else r1<-0
Mayor que (R)	sgt r1, r2, r3	if (r2 > r3) r1<-1 else r1<-0
Mayor que (I)	sgti r1, r2, #n	if (r2 > n) r1<-1 else r1<-0
Igual (R)	seq r1, r2, r3	if (r2 = r3) r1<-1 else r1<-0
Igual (I)	seqi r1, r2, n	if (r2 = n) r1<-1 else r1<-0
No igual (R)	sne r1, r2, r3	if (r2 ≠ r3) r1<-1 else r1<-0
No igual (I)	sne r1, r2, #n	if (r2 ≠ n) r1<-1 else r1<-0
Suma ®	add r1, r2, r3	r3 ← <r1> + <r2>
Suma (I)	addi r1, r2, #n	r2 ← <r1> + n
Resta (R)	sub r1, r2, r3	r3 ← <r1> - <r2>
Resta (I)	subi r1, r2, #n	r2 ← <r1> - n
Y-lógica (R)	and r1, r2, r3	r3 ← <r1> AND <r2>
Y-lógica (I)	and r1, r2, #n	r2 ← <r1> AND n
Salto si = 0	beqz	if(r1==0) pc ← dirección
Salto si ≠ 0	bnez	if(r1 ≠ 0) pc ←dirección
Bifurcación	j nombre	Pc ← dirección
Bifurcación reg.	jr r1	pc ← r1
Carga	lw	r3 ← M[r1 + dirección]
Almacenamiento	sw	M[r1 + dirección] ← r3

Arquitectura