LES INSTRUCTIONS CONCURRENTES ET SEQUENTIELLES

I. INTRODUCTION

Une description VHDL a deux domaines: un domaine séquentiel et un domaine simultané (concurrent).

Le domaine séquentiel est représenté par les process ou les sous-programmes (procedure, function), qui contiennent des instructions séquentielles. Ces instructions sont exécutées dans l'ordre dans lequel elles apparaissent dans le process ou le sous-programme, comme dans les langages de programmation tel que le C.

Le domaine concurrent est représenté par une architecture qui contient des process, des appels de sous programmes, affectations de signaux et instanciations de composants.

Dans cette partie, on présente le format et l'utilisation des instructions séquentielles et concurrentes. Comme exemples certains circuits combinatoires et séquentiels de base sont décrits, tels que des multiplexeurs, des décodeurs, des bascules, des registres, et des compteurs.

II. LES INSTRUCTIONS CONCURRENTES

Les instructions concurrentes ne peuvent existées qu'à l'extérieur de processus (process) ou à l'extérieur de sous programmes (function, procedure). Elles sont exécutées simultanément et en permanence. : On en trouve :

- Assignation concurrent d'un signal: **<=**
- Déclaration d'un process : Utilisé pour que les instructions séquentielles fassent partie du traitement simultané
- Appel de procedure : Un appel concurrent de procédure est un appel exécuté en dehors d'un process. C'est un appel qui se comporte comme process.
- Appel d'une fonction : C'est un appel qui se comporte comme process.
- Assignation conditionnelle : when...else (utilisée exclusivement à l'extérieur d'un process). Voir affectation sélective concurrente .
- Assignation sélective : with...select ...when...when (utilisée exclusivement à l'extérieur d'un process)
- Instanciation d'un composant : port map
- Instruction generate
- Les définitions de blocs

Exemple 1 : concurrente Assignation *(<=)*

Considérons un decodeur (démultiplexeur) 2 vers 4, l'ordre dans lequel seront écrites les instructions n'a aucune importance.

```
entity demux2_4 is
    port(
        IN0, IN1          : in   std_logic;
        D0, D1, D2, D3    : out  std_logic
    );
end demux2_4;

architecture description of demux2_4 is
begin
    D0 <=   (not(IN1) and not(IN0));          -- First instruction
    D1 <=   (not(IN1) and IN0);               -- Second instruction
    D2 <=   (IN1 and not(IN0));               -- Third instruction
    D3 <=   (IN1 and IN0);                    -- Fourth instruction
end description;
```

L'architecture ci dessous est équivalente à:

```
architecture description of demux2_4 is
begin
    D1 <= (not(IN1) and IN0);                 -- Second instruction above
    D2 <= (IN1 and not(IN0));                 -- Third instruction above
    D0 <= (not(IN1) and not(IN0));            -- First instruction above
    D3 <= (IN1 and IN0);                      -- Fourth instruction above
end description;
```

L'instruction définissant l'état de D0 à été déplacée à la troisième ligne, la synthèse de cette architecture est équivalente à la première. On aura le même circuit ci dessous pour les deux descriptions

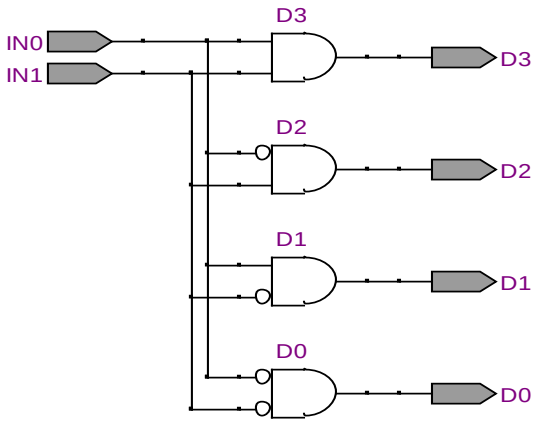

Figure III.1: Synthèse des instructions concurrentes

Exemple 2 : process Déclaration

```
REG_32: process(Clk, Clear)
```

```
begin
    if Clear='1' then
        Output <= (others=>'0');
    elsif Clk='1' then
        Output <= input after 250 ps;
    end if;
end process REG_32;
```

Exemple 3 : Call of procedure

```
architecture V1 of Var is
begin
    vector_to_int(bits, flag, number);          -- Call of procedure vector_to_int
end V1;
```

Equivalent à:

```
architecture V1 of Var is
begin
    process (bits, number);
    begin
        ……;
    end process;
end V1;
```

- Noter bien que l'appel est exécuté en dehors de tout process et qu'il est équivalent à un process.

III. LES INSTRUCTIONS SEQUENTIELLES

Les instructions séquentielles apparaissent uniquement à l'intérieur des processus (process) ou des sous programmes (function, procedure). Au sein de ces descriptions, *les instructions sont exécutées de manière séquentielle, l'une après l'autre*. L'ordre des déclarations est donc important. On en trouve :

- Assignation séquentiel du signal : **<=** (signal)
- Assignation de varaibles : **:=** (variable)
- Assignation conditionnelle : if …then…elsif…then…else…end if;
- Assignation sélective : case…is…when…=>…when…=>…end case;
- Boucle : loop;
- Boucle : for … in … loop…end loop;
- Boucle : while… loop…end loop;
- Instructions : next et exit
- Instruction : assert

IV. AFFECTATIONS STRUCTUREES

IV. 1. AFFECTATIONS SELECTIVES CONCURRENTES

Ces instructions doivent existées à l'extérieur de tout process.

IV.1.1. L'AFFECTATION SELECTIVE CONCURRENTE with select <= when

L'affectation sélective with select <= when, aiguille, à l'aide d'une expression de sélection (selecteur), le résultat de plusieurs expressions vers un signal cible.

- C'est une instruction concurrente *à utiliser exclusivement en mode combinatoire à l'extérieur d'un* process : si on essaye d'insérer la structure de séclection with...select...when à l'intérieur d'un process, on obtiendra le message d'erreur *Illegal sequential statement*.

Syntaxe

```
with selecteur select
    signal_cible <= affectation_1          when   v1_selecteur,
                    affectation_2          when   v2_selecteur,
                    ..........................................
                    affectation_n          when   vN_selecteur,
                    affectation_default    when   others;
```

Avec :

- affectation_1 est la valeur que prendrait le signal_cible si la valeur du selecteur est v1_selecteur, affectation_2 est la valeur que prendrait le signal_cible si la valeur du Selecteur est v2_selecteur, etc....
- affectation_i et vi_selecteur peuvent être des expressions logiques aussi complexe quelles soient, pour vu qu'elles donnent un résultat conforme avec le type attendu.

- Si le nombre de configuration possible du sélecteur est plus grand que le nombre réellement considéré par les instructions when, la clause when others devrait être utilisé pour considérer tous les cas non considérés par l'instruction when. Ceci permet d'éviter la création de *latch* par accident.
- L'affectation sélective whith select <= when peut être toujours ramenée à l'affectation sélective séquentielle case is when (voir plus loin). Les différences entre les deux affectations sont :
  - La première (whith select <= when) produit un process implicite sensible au Selecteur ; alors que dans la deuxième (case is when) le process est produit d'une manière explicite.
  - Dans la première, les instructions sont concurrentes. Elles devraient se trouvées donc *à l'extérieur* d'un process. Dans la deuxième forme (case is when), les instructions sont séquentielles et elles devraient se trouvées par conséquent *à l'intérieur* d'un process.

Exemple 1: Développement d'un décodeur 2/4

La figure ci-dessous représente un décodeur 2/4. Le principe de fonctionnement de ce décodeur est les suivant :

Principe :

- L'entrée Sel=(Sel(1), Sel(0)) permet de sélectionner laquelle des quatre sortie Output=(Output(3), Output(2), Output(1), Output(0)) est à activer.

Figure III.2: Schéma d'un décodeur 2/4

Dans la mise en œuvre ci-dessous, il ne sera pris en compte que l'entrée de sélection.

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

entity Deco_2_4 is
    port(
        Sel     : in    std_logic_vector(1 downto 0);
        Output  : out   std_logic_vector(3 downto 0)
    );
end deco_2_4;

architecture Arch of Deco_2_4 is
begin
    with Sel select
        Output <= "0001"  when "00",
                  "0010"  when "01",
                  "0100"  when "10",
                  "1000"  when "11",
                  (others =>'-') when others;
end Arch;                                   -- The minus '-' of key 6, means don't care
```

Noter bien

- *Noter la présence de l'affectation par défaut* when others: En simulation, le type std_logic comporte plus de deux états possibles, ce qui conduit à un nombre de combinaison du sélecteur qui dépasse de très loin quatre cas.
- L'affectation sélective est le moyen le plus adéquat de construire un opérateur en retranscrivant sa table de vérité ;
- Si le sélecteur est une expression, celle-ci doit nécessairement être qualifiée ;

- La synthèse fait généralement appel à un ensemble de multiplexeurs dont le nombre est fonction du nombre de choix offerts par l'instructon when.

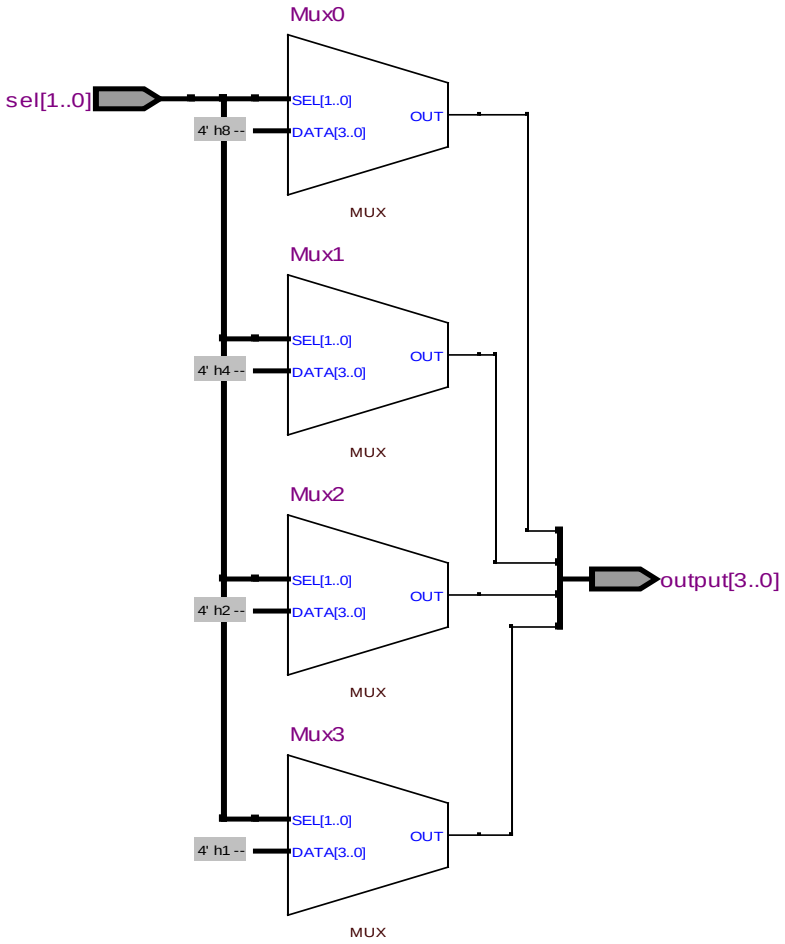

Figure III.3: Synthèse d'un décodeur 2/4 par utilisation de l'instruction with …select…when

IV.1.2. L'AFFECTATION SELECTIVE CONCURRENTE <= when else

C'est l'équivalent du test if then else du monde séquentiel: un signal est affecté de plusieurs expressions sélectives selon le résultat d'une expression de choix.

Syntaxe

```
Signal_destination   <=  source_1   when expression_1 else
                         source_2   when expression_2 else
                         ..........................................
                         source_k   when expression_k else
                         (others => default_expression);
```

expression_i est toute expression logique donnant lieu à un résultat booléen (false ou true):

- L'instruction de sélection when else *est à utiliser exclusivement en mode combinatoire* à l'extérieur d'un process **:** si on essaye d'insérer la structure de sélection when...else à l'intérieur d'un process, une erreur est signalée.
- Il devrait y avoir une instruction else inconditionnelle finale.

Exemple : Behavioral multiplexer with with selct when

Proposons-nous de développer un mumltiplexeyr 4/1, par utilisation du with selct when:

```
architecture Arch of mMux4_1 is
begin
    Data_ out  <=  Data_in0   when   Sel = "00" else
                   Data_in1   when   Sel = "01" else
                   Data_in2   when   Sel = "10" else
                   Data_in3   when   Sel = "11" else
                   (others => 'x');
end Arch;
```

Le résultat de la synthèse est donné ci dessous.

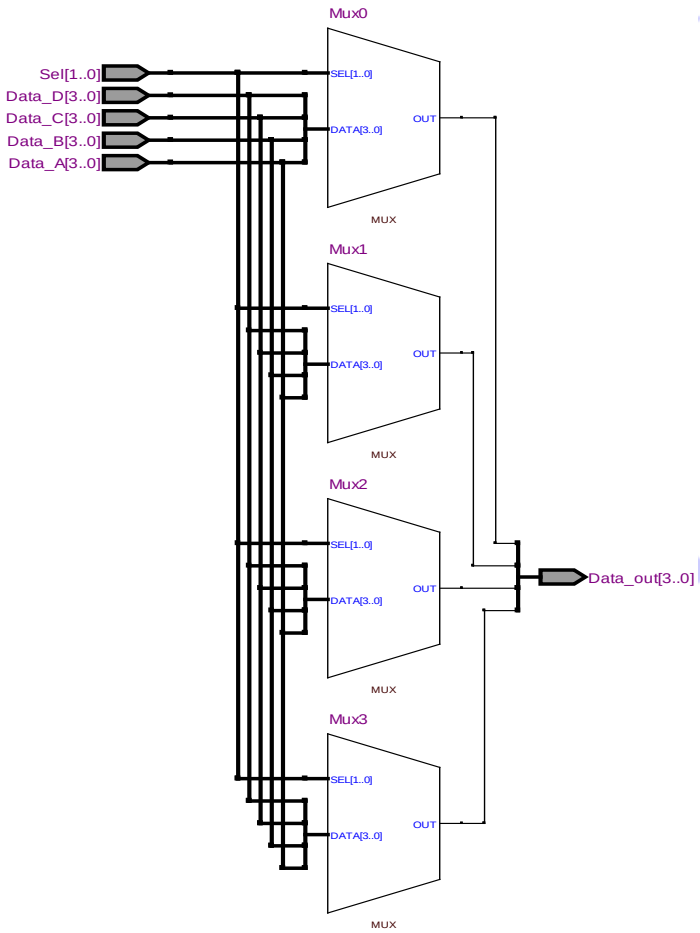

Figure III.4: Synthèse d'un Multiplexeur 4/1 par utilisation de l'instruction when …else

IV. 2. AFFECTATIONS SELECTIVES SEQUENTIELLES

Ces instructions doivent existées à l'intérieur d'un process, une function ou une procedure.

IV.2.1. L'INSTRUCTION case is when =>

L'affectation sélective séquentielle case is when en VHDL est exactement équivalente à l'affectation sélective switch en C.

- *Cette instruction est très utile dans les machines d'états.*
- C'est une instruction à utiliser exclusivement à l'intérieur d'un process, une function ou une procedure.

Syntaxe

```
Label: case selecteur  is
    when selecteur_expression_1 =>
            statement_sequence_1;
    when selecteur_expression_2 =>
            statement_sequence_2;
    when selecteur_expression_3 =>
             statement_sequence_3;
    ..........................................................
    when selecteur_expression_n =>
            statement_sequence_n;
    when others =>
            default_expression;
end case Label;
```

Où:

- selecteur_expression_i : est une ou combinaison de valeurs que peut avoir le selecteur (val1 | val2 | … | valn etc)
- statement_sequence_i : peut être un ensemble d'instructions (et non seulement une instruction).

Si la valeur du selecteur est égale à l'une des valeurs présentes selecteur_expression_i, c'est la séquence statement_sequence_i qui suit le => qui sera exécutée.

IV.2.2. L'INSTRUCTION if then elsif then else

C'est l'équivalent de l'instruction if elseif else, bien familière des langages de programmation évolué tel que le C. Elle est utilisée en VHDL pour tester plusieurs conditions, et elle est à utiliser exclusivement à l'intérieur d'un process, ou sous programmes (function, procedure). Pour un fonctionnement similaire à l'extérieur d'un process, on peut utiliser l'instruction when.

Syntaxe

```
if Logical_Expression_1 then
  Sequential Statements_1
elsif Logical_Expression_2 then
  Sequential Statements_2
elsif Logical_Expression_3 then
  Sequential Statements_3
else
  Sequential Statements
end if;
```

Où:

Logical_Expression_i: est une expression logique booleane qui devrait être évaluer à vraie (true) ou Faux (false)

Sequential Statements: est un ensemble d'instructions.

- L'imbrication des instructions if est autorisée;
- Chaque instruction « if » peut être accompagnée d'un ou plusieurs instruction « elsif » mais d'une seule instruction « else » qui temine le test;
- Chaque « Logical_Expression_i » est *une expression boolean* évaluée à true ou false.
- Le test débute avec l'instruction « if ». Si l'expression logique correspondante est évaluée à true, l'ensemble des instructions correspondantes (Sequential Statements_1) est exécutée. Si elle est évaluée à false, l'Logical_Expression_2 correspondate au « elsif » suivant est évalué et ainsi de suite jusqu'à atteindre le « else » s'il est présent.

Exemple : Behavioral multiplexer with if then elsif then else

Considérons un multiplexeur 4/1 et développons le code VHDL par untilisation de l'exécution sélective if then elsif then else. L'entity étant évidente, interesspons nous à la partie architecture.

```
architecture Arch of mux_4to1 is
begin
  process (Data_A, Data_B, Data_C, Data_D, Sel)
  begin
```

```
    if Sel ="00" then
        Data_Out <= Data_A;
    elsif Sel ="01" then
        Data_Out <= Data_B;
    elsif sel ="10" then
        Data_Out <= Data_C;
    else
        Data_Out <= Data_D;
    end if;
  end process;
end Arch;
```

La synthèse de ce programme donne le circuit ci-dessous.

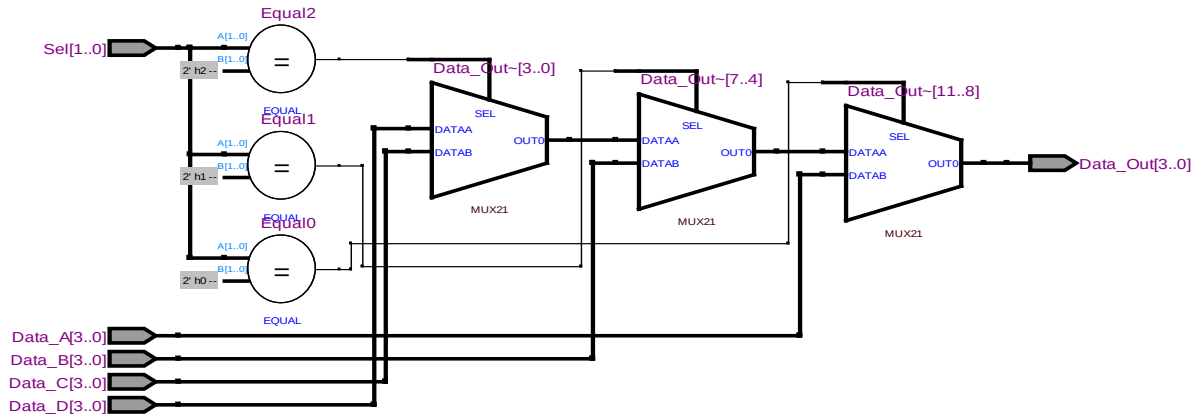

Figure III.6: Synthèse d'un Multiplexeur 4 to 1 par utilisation de l'instruction if…else

V. LES BOUCLES

V.1. LA BOUCLE for .... loop

Pour l'exécution répétée d'une séquence d'instructions un nombre fixe de fois, l'instruction for loop peut être utilisée. Elle utilise les deux mots clé for et loop.

- L'instruction exit permet de sortir de la boucle à n'importe quel moment;
- L'instruction next permet de passer à l'itération suivante.

- C'est une boucle à utiliser exclusivement dans un process, une function ou une procedure

Syntaxe

```
[ etiquette : ]  for variable  in valeur_debut to valeur_fin loop    -- Loop variable "variable" may not be declared
    Séquence d'instructions séquentielles;
end loop [ etiquette ];
```

Exemple :

Compteur du nombre de bit égal à 1 dans un vecteur avec spécification explicite de l'index du bas vers le haut.

```
library ieee;
use ieee.numeric_bit.all;

entity Count_ones is
    port (
        V       : in    bit_vector (15 downto 0);

        Count : out   signed (3 downto 0)
    );
end Count_ones;

architecture Arch of Count_ones is
begin
    process (V)
        variable Result: signed (3 downto 0);
    begin
        Result := (others => '0');
        for i in 0 to 15 loop
            if V(i) = '1' then
                Result := Result + 1;
            end if;
        end loop;

        Count <= Result;

    end process;
end Arch;
```

V.2. LA BOUCLE while … loop

Syntaxe

```
[etiquette : ] while condition loop

    Séquence d'instructions séquentielles;

end loop [etiquette ];
```

- C'est une boucle à utiliser exclusivement dans les process, les function ou les procedure

La condition est testée avant chaque exécution de la boucle. Si la condition est true, la séquence d'instructions dans le corps de la boucle est exécutée, après quoi le contrôle est transféré au début de la boucle. Si la condition est false, la séquence d'instructions dans le corps de la boucle est sautée et l'instruction après le mot clé end loop (clause de fin de boucle) est exécutée.

Exemple : 1 of 4 Decoder

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

entity Deco_1_4 is
    port(
        Sel       : in    std_logic_vector (1 downto 0);

        Output   : out std_logic _vector (3 downto 0)
    );
end Deco_1_4;
```

```
architecture Arch of Deco_1_4 is
begin
    process (Sel)
        variable  i  :  integer range 0 to 4;          -- declare an integer variable "i" varying from 0 to 4
    begin
        Output <= (others => '0');                     -- set the output to X'0 '
        i := 0;                                        -- initializes the loop variable "i"
        while (i <= 3) loop                            -- start of the loop with the output condition
            if (to_integer(unsigned(Sel)) = i) then    -- examines which output must be activated
                Output (i) <= '1';                     -- activates the corresponding output
            end if;
            i := i + 1;                                -- next iteration
        end loop;                                      -- end of the loop
    end process;
end Arch;
```

La boucle définit un décodeur : Parmi N sorties, une seule est activée. C'est celle dont l'ordre est donné par l'entrée de sélection Sel.

- Les boucles while loop peuvent être imbriquées

```
E1: while i < 10 loop
    E2: while j < 20 loop
    ...
    end loop E2;
end loop E1;
```

V.3. LA BOUCLE loop

loop est une instruction de bouclage. Elle peut s'utiliser *seule* ou avec for ou while, comme on l'a déjà vu ci-dessus. Il est question, dans ce paragraphe, de considérer cette instruction seule, pour répéter une série d'instructions.

- C'est une boucle à utiliser exclusivement dans les process, les function ou les procedure
- La sortie de la boucle est signalée explicitement par l'instruction exit. En absence de l'instruction exit, la boucle est exécutée indéfiniment.
- Le passage à l'itération suivante est signalé par l'instruction next.

Syntaxe

```
Label : loop
    Séquence d'instructions séquentielles;
    exit when expression ;
end loop;
```

Exemple : 1 of 4 Decoder

Reprenons l'exmple du décodeur déjà traité avec la boucle while ci-dessus. Le code VHDL qui traduit le fonctionnement de ce décodeur, par utilisation de l'instruction loop, est donné ci-dessous.

```
library ieee;
use ieee.std_logic_1164.all;
entity Deco_1_4 is
    port(
        Sel        : in    std_logic_vector (1 downto 0);
        Output   : out std_logic _vector (3 downto 0)
    );
end Deco_1_4;
architecture Arch of Deco_1_4 is
begin
    process (Sel)
        variable  i  :  integer range 0 to 4;
    begin
        Output <= (others => '0');
        i := 0;
        loop                                        -- Start of the loop
            if (to_integer(unsigned(Sel)) = i) then -- examines which output must be activated
                Output (i) <= '1';                  -- activates the corresponding output
            end if;
            i := i + 1;                             -- next iteration
            exit when i = 4;                        -- the output condition
        end loop;                                   -- end of the loop
    end process;
end Arch;
```

☞Notez que la sortie de la boucle est exécutée à travers l'instruction exit quand la variable "i" atteint la valeur 4.

V.4. L'INSTRUCTION next

C'est une instruction séquentielle. Elle se combine avec toutes les boucles for, while et loop pour passer à l'itération suivante.

Syntaxe

```
next [loop_label] [when condition];
```

Exemples :

```
next;                                   -- executes the next iteration unconditionally without
                                        -- branch label
next outer_loop;                        -- executes the next iteration unconditionally with the
                                        -- branch label
next when A>B;                          -- executes the next iteration conditionally without the
```

```
                                                    -- branch label
next this_loop when C=D or done;                    -- executes the next iteration conditionally with the
                                                    -- branch label."done" is Boolean variable
```

V.5. L'INSTRUCTION exit

C'est une instruction séquentielle. Elle se combine avec toutes les boucles for, while et loop pour sortir d'une boucle.

Syntaxe

```
exit [loop_label] [when condition];
```

Exemple :

```
L2: loop
    A:= A+1;
exit L2 when A > 10;                    -- The loop output is executed when the value
end loop L2;                            -- of the variable A exceeds 10
```