Annexes

Chapitre II

1- Une présentation du langage UML[1].

Le langage UML regroupe plusieurs diagrammes qui permettent de modéliser les différents aspects des systèmes considérés. Nous ne présentons ici que les deux diagrammes que nous utilisons.

1-1. Le diagramme statique de classes.

Le diagramme statique de classes permet de représenter l'ensemble des classes qui interviennent dans un système, leur hiérarchisation et leur interconnexion. Il permet une description avec la même notation pendant les phases d'analyse, d'analyse détaillée et de conception.

Une classe a un nom et des membres, ces membres peuvent être des attributs ou des méthodes typées. On peut représenter une classe sous sa forme explicite (cf. Figure 1) en faisant apparaître tous ses membres, ou sa forme condensée (cf. Figure 2) en ne faisant apparaître que son nom. Dans la suite nous les représenterons par défaut sous leur forme condensée.

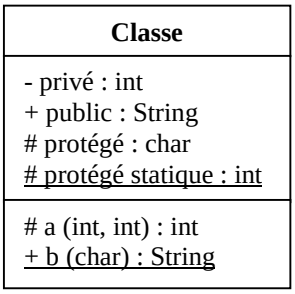

Figure 1- Classe sous sa forme explicite.

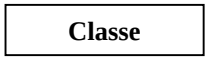

Figure 2 – Classe sous sa forme condensée.

Figure 3 – Classe abstraite.

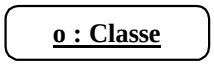

Figure 4 – Objet 'o' instancié à partir de la classe Classe.

Une classe peut être concrète, des objets (cf. Figure 4) pourront alors être instanciés à partir de cette classe. Elle peut aussi être abstraite (cf. Figure 3), elle ne peut alors pas être instanciée. Les membres d'une classe concrète ont un qualifieur d'accès qui peut être public (+)[2], privé (-), protégé (#) ou statique (souligné). Ces qualifieurs précisent les conditions sous lesquelles les membres sont accessibles à partir d'une classe. Tous les membres d'une classe abstraite sont publics.

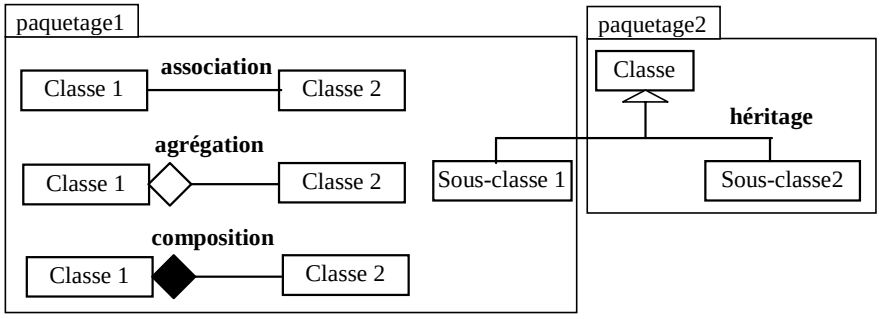

Figure 5 - Modèle statique

[1] UML : Unified Modelling Language. Langage de modélisation qui veut unifier Rumbaugh, Booch et Jacobson.
[2] Pour des raisons de commodités d'écriture, les membres publics sont parfois notés sans qualifieur.

Plusieurs classes peuvent être regroupées dans des paquetages, leur comportement interfère par l'intermédiaire de relations : l'association, l'agrégation, la composition et l'héritage (cf. Figure 5).

La composition dénote une responsabilité de la classe composée vis-à-vis de ses composantes. Elle est utilisée, si l'objet instancié à partir de la classe composante est spécifique à l'objet de la classe composée et ne pourra pas être partagé avec d'autres instances de la classe composée. Fréquemment, on considère que c'est la classe composée qui doit instancier et détruire personnellement ses classes composantes.

1-2. Le diagramme de séquences.

Un diagramme de séquences (cf. Figure 6) modélise les séquences ou scénarios des communications entre classes dans un cas particulier afin de réaliser une opération plus complexe. Ils mettent en œuvre les objets et les acteurs qui participent à la communication et sont représentés par leur nom et une barre verticale. Les objets sont représentés par leur classe dans un diagramme statique associé. Des flèches partent de la barre qui représente l'objet qui déclenche une communication et pointent en direction de la barre qui représente l'objet avec lequel la communication est établie. La flèche est nommée par le nom et les arguments de la méthode appelée pour réaliser la communication.

Une flèche suppose d'une part qu'il existe une relation dans le diagramme statique entre les deux intervenants, et d'autres part que la méthode qui correspond à la flèche soit déclarée dans la classe de l'objet destination et accessible à partir de l'objet source.

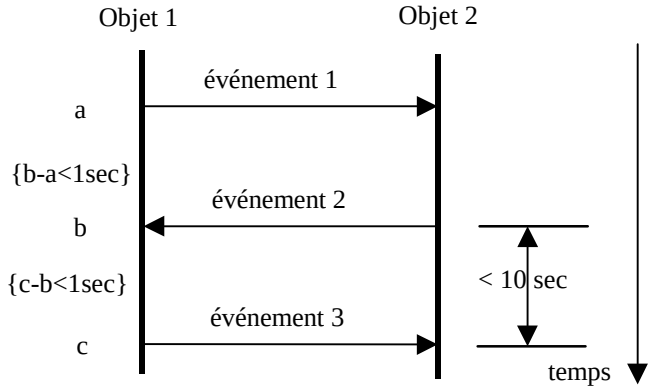

Figure 6 – Diagramme de séquences.

Chapitre III

2- Composants générés en utilisant le méta-modèle.

2-1. Le composant *ROM*.

```
// Fichier généré.
// à partir du ServiceProvider component.model.memory
package component.basic;

import sep.model.*;
import sep.model.event.*;
import sep.type.*;

public class ROM extends EdgeComponent {
  protected component.model.memory provider=new component.model.memory();
  protected Port adr;
  protected Port out = new Port();

  public void read() {
   if (isReset())
          reset();
   else {
          int arg0= ((IntValue)adr.read().castTo(new IntValue(0)))
                                                    .getIntValue();
          getOUTPort().emit(provider.read(arg0));
          stateChanged(new StateEvent(this, "read"));
   }
  }

  public void setREADPort(Port p) {
   p.addRisingEdgeListener(new EventAdapter() {
          public MaterialComponent getComponent() { return ROM.this; }
          public void eventOccurs() {
                read();
          }
   });
  }

  public void setADRPort(Port p) {  adr=p; }
  public Port getOUTPort() {  return out; }
// ----- Parametres -----
  public void setFile(java.io.File p0) { provider.setFile(p0); }
  public java.io.File getFile () { return provider.getFile(); }

  public void setSize(int p0) { provider.setSize(p0); }
  public int getSize () { return provider.getSize(); }

  public void setBaseAddress(int p0) { provider.setBaseAddress(p0); }
  public int getBaseAddress () { return provider.getBaseAddress(); }
// ----- Fin parametres -----
  public void reset() { out.reset(); }
}
```

2-2. Le composant *RAM*.

```
// Fichier généré.
// extension du composant ROM
package component.basic;

import sep.model.*;
import sep.model.event.*;
import sep.type.*;

public class RAM extends ROM {
  protected Port in;

// Hérite du service read
```

```
  public void write() {
   if (isReset())
          reset();
   else {
          int arg0= ((IntValue)adr.read().castTo(new IntValue(0)))
                                                              .getIntValue();
          sep.type.Value arg1 = in.read();
          provider.write(arg0, arg1);
          stateChanged(new StateEvent(this, "write"));
   }
  }

  public void setWRITEPort(Port p) {
   p.addRisingEdgeListener(new EventAdapter() {
          public MaterialComponent getComponent() { return RAM.this; }
          public int getPriority() { return -1; }
          public void eventOccurs() {
                write();
          }
   });
  }

  public void setINPort(Port p) { in=p; }
}
```

2-3. Le composant *RegisterBank*.

```
// Fichier généré.
// à partir du ServiceProvider component.model.memory
package component.basic;

import sep.model.*;
import sep.model.event.*;
import sep.type.*;

public class RegisterBank extends EdgeComponent {
  protected component.model.memory provider=new component.model.memory();
  protected BankPort adr = new BankPort();
  protected Port in;
  protected BankPort out = new BankPort();

  public RegisterBank() {
   adr.setSize(2);
   out.setSize(2);
  }

  public void write() {
   if (isReset())
          reset();
   else {
          int arg0= ((IntValue)adr.read().castTo(new IntValue(0)))
                                                        .getIntValue();
          sep.type.Value arg1 = in.read();
          provider.write(arg0, arg1);
          stateChanged(new StateEvent(this, "write"));
   }
  }

  public void read() {
   if (isReset())
          reset();
   else {
          for(int i=0; i<2; i++)
                read(i);
   }
  }
```

```
  public void read(int i) {
   int arg0= ((IntValue)adr.read(i).castTo(new IntValue(0))).getIntValue();
   getOUTPort(i).emit(provider.read(arg0));
   stateChanged(new StateEvent(this, "read"));
  }

  public void setWRITEPort(Port p) {
   p.addRisingEdgeListener(new EventAdapter() {
          public MaterialComponent getComponent() { return RegisterBank.this; }
          public void eventOccurs() {
                write();
          }
   });
  }

  public void setADRPort(Port p, int i) { adr.setPort(p, i); }
  public void setINPort(Port p) { in=p; }
  public void setREADPort(Port p) {
   p.addRisingEdgeListener(new EventAdapter() {
          public MaterialComponent getComponent() { return RegisterBank.this; }
          public int getPriority() { return 1; }
          public void eventOccurs() {
                read();
          }
   });
  }

  public Port getOUTPort(int i) { return out.getPort(i); }
// ----- Parametres -----
  public void setSize(int p0) { provider.setSize(p0); }
  public int getSize () { return provider.getSize(); }
// ----- Fin parametres -----
  public void reset() { out.reset(); }
}
```

Chapitre IV *3- Grammaire de SEP-ISDL.*

programme ::= macro-definitions (schema_declaration instruction_definition) *

macro_definitions ::= (**'<'** macro_definition **'\n'** (macro_action_definition)*)*
macro_definition ::= macro_nom (element) + (**'|'** (element) +)*
element ::= identificateur | **'_NUM_'**

macro_action_definition ::= macro_action_nom **':' '\n'** (actions **'\n'**)*
actions ::= port_nom valeur type
| port_nom valeur_multiple type_multiple
valeur ::= string | **'#this'** | **'$this'**
valeur_multiple ::= valeur (**','** valeur) $^{+}$
identificateur ::= string **|** '_' macro_nom **'_'**

schema_declaration ::= **'>'** schema_nom **'('** parametre (**','** parametre)* **')'** condition **'\n'**
schema_nom ::= identificateur
parametre ::= **'bit'** | (identificateur) +
condition ::= true | **'?'** | ø

instruction_definition ::= (action (commentaire) * **'\n'**) *
action ::= action_simple | macro_action

action_simple ::= port_nom expression type
| port_nom expression_multiple type_multiple
| **'sleep'**
port_nom ::= string
expression ::= expression_simple | expression_conditionnelle
expression_simple ::= string | **'#'** integer | **'$'** integer
condition ::= **'#'** integer
expression_conditionnelle ::= condition **'?'** expression_simple **':'** expression_simple
expression_multiple ::= expression (**','** expresion) $^{+}$
type ::= **'Int'** | **'Level'** | **'String'** | **'CodeOp'** | **'?'**
type_multiple ::= **'Multiple'** | **'MultString'**

macro_action ::= integer **'.'** macro_action_nom
| integer '.' macro_action_nom condition
commentaire ::= string

Chapitre VI

4- Etude du mécanisme de liaison dynamique de Java.

```
/** fichier DoubleValue.java */
public class DoubleValue {
   private double value;
   public DoubleValue(double initV) { value = initV; }
   public DoubleValue addition(DoubleValue v) {
                System.out.println("Addition Double x Double");
// Code rajouté pour résoudre le point (2) {
                if (v instanceof IntValue)
                        return addition((IntValue)v);
// }
          return new DoubleValue(value + v.value);
   }
// Code Rajouté pour résoudre le point (3) {
      public DoubleValue addition(IntValue v) {
            System.out.println("Addition Double x Int");
            return v.addition(this);
      }
// }
   public String toString() { return "Double:"+value; }
}

// -------------------- Fichier IntValue.java ----------------------
public class IntValue extends DoubleValue {
   private int value;
   public IntValue(int initV) { super(initV); value = initV; }
      public DoubleValue addition(IntValue v) {
            System.out.println("Addition Int x int");
          return new IntValue(value + v.value);
   }
   public String toString() { return "Int:"+value; }
}

// ------------------- Fichier Operations.java -------------------------
public class Operations {
  static public void main(String args[]) {
   DoubleValue e1 = new IntValue(1),      // Type = Double, Classe=Int
                r1 = new DoubleValue(2.3), // Type = Double, Classe=Double
                e2 = new IntValue(3),        // Type = Double, Classe=Int
                r2 = new DoubleValue(4.5); // Type = Double, Classe=Double

   IntValue i = new IntValue(-5);      // Type = Int, Classe=Int

      System.out.println(i+"+"+r1+"="+     i.addition(r2)      +"\n");
      System.out.println(r1+"+"+r2+"="+    r1.addition(r2)     +"\n");
      System.out.println(r1+"+"+i+"="+     r1.addition(i)      +"\n");
      System.out.println(i+"+"+i+"="+      i.addition(i)       +"\n");
      System.out.println(i+"+"+e2+"="+     i.addition(e2)      +"\n");
      System.out.println(e1+"+"+e2+"="+    e1.addition(e2)     +"\n");
      System.out.println(e1+"+"+i+"="+     e1.addition(i)      +"\n");
 }
}

// ---------------------- RESULTATS ----------------------------
/* Première solution :

>java Operations
Addition Double x Double
Int:-5+Double:4.5=Double:-0.5

Addition Double x Double
Double:2.3+Double:4.5=Double:6.8

Addition Double x Double
Double:2.3+Int:-5=Double:-2.7
```

```
Addition Int x int
Int:-5+Int:-5=Int:-10

Addition Double x Double
Int:-5+Int:3=Double:-2.0   // 2 entiers mais opération sur les doubles, cas (2)

Addition Double x Double
Int:1+Int:3=Double:4.0     // 2 entiers mais opération sur les doubles, cas (2)

Addition Double x Double
Int:1+Int:-5=Double:-4.0  // 2 entiers mais opération sur les doubles, cas (3)
*/

/*   Deuxième   solution   :   On   rajoute   la   définition   de   DoubleValue
addition(IntValue) dans la classe DoubleValue :

>java Operations
Addition Double x Double
Int:-5+Double:4.5=Double:-0.5

Addition Double x Double
Double:2.3+Double:4.5=Double:6.8

Addition Double x Int
Addition Double x Double
Double:2.3+Int:-5=Double:-2.7

Addition Int x int
Int:-5+Int:-5=Int:-10

Addition Double x Double
Int:-5+Int:3=Double:-2.0   // 2 entiers mais opération sur les doubles, cas (2)

Addition Double x Double   // 2 entiers mais opération sur les doubles, cas (2)
Int:1+Int:3=Double:4.0

Addition Int x int
Int:1+Int:-5=Int:-4        // Cas (3) résolu.
*/

/* Solution finale : On rajoute le test dynamique de type et le cast:

>java Operations
Addition Double x Double
Int:-5+Double:4.5=Double:-0.5

Addition Double x Double
Double:2.3+Double:4.5=Double:6.8

Addition Double x Int
Addition Double x Double
Double:2.3+Int:-5=Double:-2.7

Addition Int x int
Int:-5+Int:-5=Int:-10

Addition Double x Double
Addition Int x int
Int:-5+Int:3=Int:-2          // Cas (2) résolu.

Addition Double x Double
Addition Int x int
Int:1+Int:3=Int:4            // Cas (2) résolu

Addition Int x int
Int:1+Int:-5=Int:-4
*/
```

Chapitre XI 5- Le composant BarrelShifter.

```
public class BarrelShifter implements sep.model.ServiceProvider {
   public BitValue shl(BitValue in) { return sep.type.Type.perform("shl", in); }
   public BitValue shr(BitValue in) { return sep.type.Type.perform("shr", in); }
   public BitValue shl4(BitValue in) {
          return sep.type.Type.perform("shl", in, new IntValue(4));
   }
   public BitValue shr4(BitValue in) {
          return sep.type.Type.perform("shr", in, new IntValue(4));
   }
   public BitValue rol(BitValue in) { return sep.type.Type.perform("rol", in); }
   public BitValue ror(BitValue in) { return sep.type.Type.perform("ror", in); }
   public BitValue id(BitValue in) { return (BitValue)in.getClone(); }
   public BitValue clr(BitValue in) { return new BitValue(); }
}
```

Chapitre XI 6- Le composant ZCN.

```
public class ZCN implements sep.model.ServiceProvider {
   public final int NORMAL=0, BOUCLE=1;
   protected int lcLow = -1, lcHigh [], last [], first[];
   private int current = 0;
   protected Value out = LevelValue.Undefined;

   public void rep(int n) { lcLow = n; }
   public void bkrep(int m, int last) throws RuntimeException {
          current++;
          lcHigh[current] = m;
          last[current] = last;
          first[current] = -1;
   }

   public int next(int pc) {
          if (lcLow>=0) {
                 out = new IntValue(0);
                 lcLow--;
                 return BOUCLE;
          } else if (current>=0 && lcHigh[current]>=0) {
                 if (first[current] == -1)
                        first[current] = pc;
                 else if (pc==last[current]) {
                        out = new IntValue(first[current]-last[current]);
                        if (--lcHigh[current] <0);
                               current--;
                        return BOUCLE;
                 }
                 return NORMAL;
          }
   }
   public Value output() { return out; }

// Paramètres
   public void setNestedLoop(int size) {
          lcHigh = new int [size];
          last = new int [size];
          first = new first[size];
          for (int i=0; i<size; i++)
                 lcHigh = -1;
          current = 0;
   }
   public int getNestedLoop() {
          return lcHigh==null?0:lcHigh.length;
   }
}
```

Chapitre XI *7- Codage SEP-ISDL relatif aux architectures programmables modélisées.*

7-1. Le processeur maîtrise.

cf . chapitre X.

7-2. Les processeurs PINE et OAK+.

```
<SR lt|le|eq|ge|gt

// Les registres rN sont commandés par le DAU.
<rN r0|r1|r2|r3|r4|r5|cfgi|cfgj|sp
:read
     DAUcom readGDP,#this Multiple
:write
     DAUcom write,#this Multiple

// Adressage indirect, l'adresse est lue dans un registre
<rIind (r0)|(r1)|(r2)|(r3)
:read
   DAUcom readX,#this Multiple
   memX read String
:write
   DAUcom readX,#this Multiple
   memX write String

<rJind (r4)|(r5)
:read
   DAUcom readY,#this Multiple
   memY read String
:write
   DAUcom readY,#this Multiple
   memY write String

<ind _rIind_|_rJind_

// Adressage direct : L'adresse est mise sur le bus d'adresse.
<adr _NUM_
:read
     ADR $this ?

// Adressage immédiat : donné sur le bus général.
<imm #_NUM_
:read
     GDP #this ?

// Les accumulateurs sont commandés par le module CU.
<aX a0|a1
:read
   CUcom sel,$this MultString
:write
   CUop Mov Codeop
   CUcom write,$this MultString

// Les accumulateurs sont commandés par le module CU.
<bX b0|b1
:read
   CUcom sel,$this MultString
:write
   CUBS id String // Positionne le code opération du BarrelShifter
   CUcom write,$this MultString

<axx a0h|a1h|a0l|a1l|b0h|b0l|b1h|b1l
:read
   CUcom GDP:=,$this MultString   // Positionne l'unite de saturation
                          // Et sélectionne l'entrée imm du CBU et CU
```

```
<sti st0|st1|st2
:write
   SRcom write_,$this MultString

<p p
:read
   CUcom   op2:=P  MultString

<mod +1|-1|+s|no

// L'instruction Mov.

<srcMov1 _aX_|_bX_
<srcMov2 _imm_|_rN_|_ind_|_axx_
<dstMov2 _rN_|_aX_|_bX|_sti_

>mov (srcMov1,srcMov1) true
1.read
sleep
2.write
pcCom pc:=pc+v1 String

>mov (srcMov2,dstMov2) true
1.read
sleep
2.write
pcCom pc:=pc+v1 String

>mov (srcMov2,ind,mod) true
DAUmod $3,$3 Multiple
1.read
sleep
2.write
pcCom pc:=pc+v1 String

// Instruction clr
>clr (aX) true
CUop clr CodeOp
CUcom write_,$1 MultString
pcCom pc:=pc+v1 String

>clr (bX) true
BS clr CodeOp                 // Positionne le BarrelShifter.
CUcom write_,$1 MultString
pcCom pc:=pc+v1 String

// Boucles Materielles
<srcrep imm|rN

>bkrep (srcrep) true
1.read
pcCom pc:=pc+v1 String

>bkrep_end (adr) true
1.read
SCNcom bkrep
pcCom pc:=pc+v1 String

>rep (srcrep) true
1.read
SCNcom rep
pcCom pc:=pc+v1 String

// Operations arithmetiques
<srcArit p|aX|imm
<op add|sub
```

```
>mpy (ind,ind,mod,mod) true
DAUmod $3,$4 Multiple
1.read
2.read
CUCom p:=x*y String
pcCom pc:=pc+v1 String

>mac (ind,ind,aX,mod,mod) true
DAUmod $4,$5 Multiple
1.read
2.read
CUop add Codeop
CUCom mulacc_,$3 MultString
pcCom pc:=pc+v1 String

>msub (ind,ind,aX,mod,mod) true
DAUmod $4,$5 Multiple
1.read
2.read
CUop sub Codeop
CUCom mulacc_,$3 MultString
pcCom pc:=pc+v1 String

>_op_ (srcArit,aX) true
CUcom   op1:=,$2 MultString
1.read
CUop    $0 CodeOp
CUcom   write_,$2 MultString
pcCom pc:=pc+v1 String

// Manipulation de bit
<BMUop shl|shr|shl4|shr4|rol|ror

>func (aX,BMUop) true
1.read
BS $2 String
CuCom write,$1 MultString
pcCom pc:=pc+v1 String

>func (bX,BMUop) true
1.read
BS $2 String
CuCom write,$1 MultString
pcCom pc:=pc+v1 String

>modr (ind,mod) true
DAUmod $2,$2 Multiplex
DAUcom read,#1 Multiple
pcCom pc:=pc+v1 String

// Interruptions
>int2
DAUmod no,no Multiple
DAUCom read,8 Multiple
mode int2 String
v2 500 Int
pcCom pc:=v2 String

>nmi
DAUmod no,no Multiple
DAUCom read,8 Multiple
mode nmi String
v2 600 Int
pcCom pc:=v2 String

>reti
DAUmod no,no Multiple
DAUCom read,8 Multiple
```

```
mode normal String
pcCom pc:=pc+v1 String

>nop
pcCom pc:=pc+v1 String

>? (?,?) false
pcCom pc:=pc+v1 String

>? (?) false
pcCom pc:=pc+v1 String
```

7-3. Le processeur ARM7.

```
<SR lt|le|eq|ge|gt

<reg r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14,r15
:readA
   regCom readA,#this Multiple
   op1_sel 0 Int             // alu.op1 := regA
:readB
   regCom readB,#this Multiple
   op2_sel 0 Int             // alu.op2 := regB
:write
   regCom write,#this Multiple

<imm _NUM_
:readB
   op2    #3     ?           // met la donnée sur le bus.
   op2_sel       1 Int              // alu.op2 := immediate.

<pc pc
:readB
   op2_sel 2 Int             // alu.op2 := pc.

<arit adc|add|and|bic|xor|or|sub
:exe
   cop #this CodeOp
   sleep
<src _reg_|_imm_|pc

>_arit_ (reg,reg,src) true
sr_en     HIGH   Level  // Autorise l'ecriture des registres d'etat.
2.readA
3.readB
0.exe
1.write
comPc pc:=pc+1 String
comRa ra:=pc String

>test Add|Sub|Xor|And   // Correspond aux instructions Cmn,Cmp,Teq,Tst

>_test_ (reg,dst) true
sr_en     HIGH   Level  // Autorise l'ecriture des registres d'etat.
1.readA
2.readB
cop $0 CodeOp
comPc pc:=pc+1 String
comRa ra:=pc String

// Les instructions mov
>mov (reg,src) true
sr_en     HIGH   Level
2.readB
copMov    CodeOp  // alu.out := alu.op2
sleep
1.write
comPc pc:=pc+1 String
```

```
comRa ra:=pc String

>mov (pc,reg) true
sr_en     HIGH   Level
2.readB
copMov    CodeOp  // alu.out := alu.op2
comPc pc:=alu String
comRa ra:=pc String

// Les branchements
>b (imm) true
1.readB
cop Mov    CodeOp  // alu.out := alu.op2
comPc pc:=ALU String
comRa ra:=pc String

// Branchement et echange reg:=pc et pc:=(reg)
>bx (reg) true
selWrite 1 Int    // reg.in := pc
1.readB
cop Mov    Codeop // alu.out := alu.op2
sleep
1.write
comPc pc:=ALU String
comRa ra:=pc String
selWrite 0 Int    // reg.in := ALU

// Branchement et lien r14:=pc et pc:=imm
>bl (imm) true
selWrite 1 Int    // reg.in := pc
1.read
sleep
regCom write,14 Multiple
cop Mov    CodeOp  // alu.out := alu.op2
comPc pc:=ALU String
comRa ra:=pc String
selWrite 0 Int    // reg.in := ALU

// Les interruptions
<intCode IRQ|FIQ
:vect
   op2     #this?40:80  Int
   op2_sel 1    Int          // alu.op2 := imm
   cop    Mov    Codeop       // alu.out := alu.op2

>_intCode_
M    $0  String
selWrite 1 Int   // reg.in := pc
regCom write,14 Multiple
1.vect
comPc pc:=ALU String
comRa ra:=pc String
selWrite 0 Int   // reg.in := ALU

>reti
regCom readB,14 Multiple
op2_sel 0 Int // alu.op2 := B
cop mov Codeop
comPc pc:=ALU String
comRa ra:=pc String
M    NORMAL String

// Instructions str et ldr
>str (reg,reg,dst,bit,bit)
2.readA
3.readB
cop #4?Add:Sub  Codeop
comRa ra:=ALU String
```

```
2.write #5
1.readB
mem write String
comPc pc:=pc+1
comRa ra:=pc String

>ldr (reg,reg,dst,bit,bit)
2.readA
3.readB
cop #4?Add:Sub  Codeop
sleep
comRA ra:=ALU String
mem read String
cop Mov   Codeop
1.write #5
comPc pc:=pc+1 String
comRa ra:=pc String

// Mauvaises instructions.
>?  (reg,reg,?) false
comPc pc:=pc+1 String
comRa ra:=pc String

>? (reg,?)    false
comPc pc:=pc+1 String
comRa ra:=pc String

>? (?) false
comPc pc:=pc+1 String
comRa ra:=pc String
```

8- Exemples de code assembleur.

```
   ; FFT pour le OAK+
   mov #2000,r0
   mov #2002,r1
   mov #1000,r4
   mov #0x03,st2
   mov #0x01,st1
   mov #0x0
   mov #0x03,cfgi
   mov #0x03,cfgj
   clr a0
   clr a1

   bkrep #1              // debut de la boucle, passage horizontale
   bkrep_end 29

   bkrep #1              // debut de la boucle, passage verticale des groupes
   bkrep_end 29

   bkrep #1              // debut de la boucle, passage verticale d'un papillon
   bkrep_end 28

   mov (r0),a0,NO              // debut du papillon
   mov (r0),a1,NO
   shl a1
   mpy (r1),(r4),+1,+1         // initialisation du pipeline
   mac (r1),(r4),a0,NO,-1      // calcul de la partie reelle
   msub (r1),(r4),a0,-1,+1
   sub a1,a0
   mov a1l,r2
   mov a0l,(r0),+1

   mov (r0),a0,NO              // debut du calcul des parties imaginaires
   mov (r0),a1,NO
   shl a1
   mac (r1),(r4),a0,NO,+1      // calcul de la partie imaginaire
   mac (r1),(r4),a0,NO,NO
   sub a1,a0
   mov a0l,(r0),+s             // fin du papillon et changement de groupe : +s
   mov r2,(r1),+1
   mov a1l,(r1),+s             // fin du papillon
   // incrémentation du pas horizontale
//-----------------------------------------------------
   // ----- code FIR pour le OAK+ -----
   // b(i) = Somme (k(i)*x(i))
   mov #0x211,r1   // les b(i) sont dans la mémoire X à partir de 0x211
   mov #1000,r4    // les k(i) sont dans la mémoire Y à partir de 0x1000
   mov #500,r2     // les a(i) sont dans la mémoire X à partir de 0x500

   mov #0x10,st2    // On autorise le comptage modulo pour le registre r4
   mov #0x0381,cfgj // Step=1 modulo 8

   bkrep #9   // Pour les 10 données
   bkrep_end 15
   mov r2,r0
   clr a0
   mpy (r0),(r4),+1,+1    // Amorce le pipeline
   rep #6                 // Pour tous les coefficients
   mac (r0),(r4),a0,+1,+1
   add p,a0         // Vide le pipeline
   mov a0h,(r1),+1        // b(i)H => memX
   mov a0l,(r1),+1        // b(i)L => memX
   modr (r2),+1    // Donnée suivante.

   halt
```

Chapitre XII

9- Code Esterel du contrôleur d'arrosage automatique.

Il y a trois modules, le module principal SPRINKLER qui utilise les deux modules MIXING et WATERING pour respectivement contrôler le mélange et contrôler l'arrosage.

sprinkler.strl :

```
module SPRINKLER:

input START(integer);
input MINUTE, LITRE, LOW_LEVEL, HIGH_LEVEL;

output PUMP_ON, PUMP_OFF, WV_ON, WV_OFF, FV_ON, FV_OFF;
output DONE(integer);

return R_PARAM;
return R_REPORT;

sensor HUM_S(integer), CONC_S(integer);

task PARAM(integer, integer)(integer);
task REPORT()(integer, integer);

signal HUM_REF(integer), CONC_REF(integer) in

  var X, Y, NUMBER : integer in
    loop
      % Set initial state
      emit PUMP_OFF;
      emit WV_OFF;
      emit FV_OFF;
      await START;
      NUMBER:=?START;
      exec PARAM(X,Y)(NUMBER) return R_PARAM;
    % returns references for concentration and humidity
    % associated with ?START in some data base
      weak abort
          emit CONC_REF(X);
          run MIXING[constant 2/delta]
          ||
          emit HUM_REF(Y);
          run WATERING
      when DONE;
      exec REPORT()(NUMBER, ?DONE) return R_REPORT
    % Reports water consumption for ?START
    end %loop
  end %var
end %signal

. %SPRINKLER
```

mixing.strl:

```
module MIXING:

input MINUTE, CONC_REF(integer), LOW_LEVEL, HIGH_LEVEL;

output FV_ON, FV_OFF, WV_ON, WV_OFF;

sensor CONC_S(integer);

constant delta : integer;

var REFERENCE : integer in
%Level control
  loop
    await LOW_LEVEL;
    emit WV_ON;
```

```
      await HIGH_LEVEL;
      emit WV_OFF;
    end % loop level control

  ||

  %Concentration control
    loop
      await immediate CONC_REF;
      REFERENCE:=?CONC_REF;
      every MINUTE do
        if (REFERENCE-delta)>?CONC_S
        then emit FV_ON
        else
            if (REFERENCE+delta)<?CONC_S
            then emit FV_OFF
            end %if
        end %if
      end % every concentration control
    end % loop
  end %var

  . %end MIXING
```

watering.strl:

```
  module WATERING:

  input MINUTE, HUM_REF(integer), LITRE;

  output PUMP_ON, PUMP_OFF, DONE(integer);

  sensor HUM_S(integer);

  signal CONSUMPTION(integer) in
    var REFERENCE:integer in
      emit PUMP_ON;
      trap SPRAY in
        await immediate HUM_REF;
        REFERENCE:=?HUM_REF;
        every MINUTE do
            if REFERENCE < ?HUM_S
            then
               emit PUMP_OFF;
               emit DONE(?CONSUMPTION);
               exit SPRAY
            end % if
        end % every MINUTE

      ||

        var QUANTITY:=0 : integer in
            every LITRE do
               QUANTITY:=QUANTITY+1;
               emit CONSUMPTION(QUANTITY);
            end % every LITRE
        end % var QUANTITY
      end % trap SPRAY;
    end % var REFERENCE
  end % signal CONSUMPTION
```

. %end WATERING