4. Disseny principal del programa.

En aquest capítol, com ja s'ha comentat al capítol 1 es van a donar les idees plantejades i que s'han dut a terme per a l'elaboració del present projecte.

En primer lloc cal dir que el que s'ha buscat és la millor manera de comprensibilitat del funcionament de l'arquitectura MIPS unicicle per part de l'usuari que execute el simulador.

4.1. Arquitectura MIPS unicicle.

Com ja és sobradament conegut, la simulació es correspon amb l'arquitectura mostrada a la figura 4.1, és a dir, l'arquitectura MIPS unicicle.

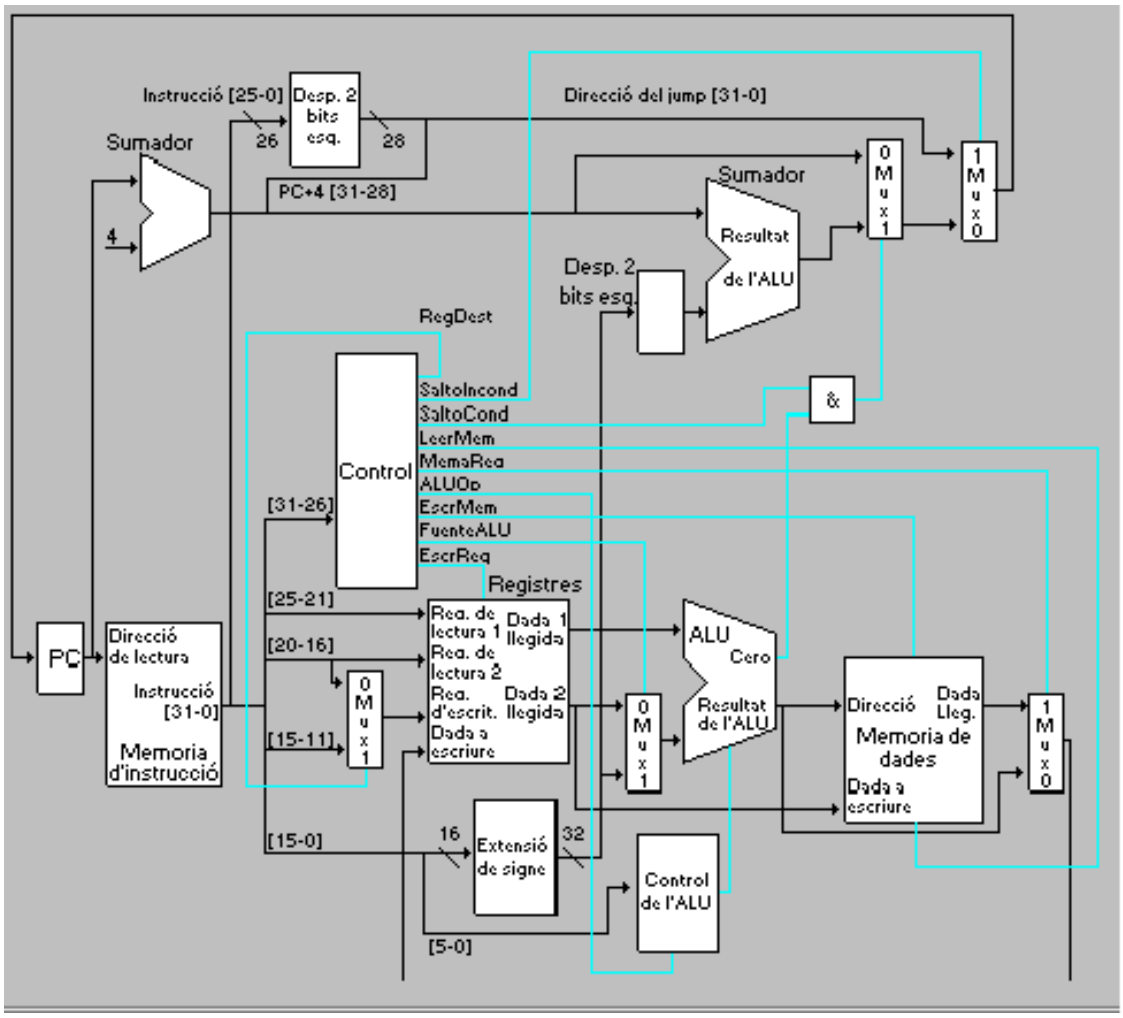


Figura 4.1.

Per visualitzar correctament la simulació d'aquesta arquitectura, és a dir, per donar una idea de visualització per part del usuari de com funciona la màquina, el que s'ha pensat ha sigut canviar de color les linees que en eixe moment (pas) estiga en funcionament, a més si conté algun valor, aquest també serà mostrat.

Encara que com hem dit la arquitectura és de tipus unicicle i açò suposa, com ja s'ha dit a la introducció, que l'execució sencera de la instrucció es realitza en tan sols un cicle de rellotge, per donar-li un caràcter més de simulació al programa, s'ha aconseguit partir les instruccions en parts, a partir d'ara s'anomenaran passos. Com es veu, açò suposa un allunyament de la realitat però, com es vora a la fi aquesta aposta aportarà una millor comprensió del camí de dades pel que passa la instrucció.

4.2. Elaboració del gràfic principal.

Com que tot el codi del programa s'ha hagut de realitzat mitjançant l'entorn de programació Borland C++ Builder, no s'ha fet ús de cap mapa de bits (arxius d'extensió bmp) per a l'elaboració del d'aquest gràfic, encara que també l'ús d'arxius d'aquest tipus haguera sigut possible, però no eficaç, per la feina que hagués comportat a l'hora de copiar com si fos una fotografia. Aquesta opció d'arxius del tipus *bmp* haguera sigut bona si per exemple realitzàrem la instrucció d'una sola vegada.

Descartada aquesta proposta, es va decidir de dibuixar l'arquitectura amb comandes del llenguatge C++, per tant ara el que s'havia de decidir era com dibuixar per una part els cuadres dels blocs funcionals de l'arquitectura i per altra les línies.

Els blocs funcionals s'han realitzat utilitzant la funció *Polygon* de l'objecte *Canvas*. Com exemple es mostrarà la realització d'un dels blocs, el corresponent a la memòria d'instruccions:

```
MemIns[0].x=a+30;
MemIns[0].y=b;
MemIns[1].x=MemIns[0].x+50;
MemIns[1].y=MemIns[0].y;
MemIns[2].x=MemIns[1].x;
MemIns[2].y=MemIns[1].y+70;
MemIns[3].x=MemIns[0].x;
MemIns[3].y=MemIns[2].y;

Canvas->Polygon(MemIns,3);
```

A d'aquesta funció com es veu, hem de passar-li dos paràmetres. Un es una estructura de tipus *POINT*, que com el seu nom ho indica, es una estructura amb dos camps de nombres enters corresponent a un punt dins del *Canvas*; tal com es veu, aquest paràmetre es un vector de quatre components, que justament són les quatre vèrtex d'un rectangle. El segon paràmetre que es passa a la funció polygon es el valor de la última component del vector que violem unir els punts, com es veu és tres. Per a que tot açò comentat funcione com s'ha comentat hem de donar valors als quatre punts que volem unir, tal com es fa abans de la crida a la funció.

Aquesta anterior fórmula s'ha utilitzat per realitzar tots els blocs funcionals. A més a més, abans de realitzar la crida a la funció, s'ha de tindre en compte dos paràmetres:

- **El color de fons dels blocs:** ja que es pot triar el color de l'àrea que queda tancada pels punts que li passem a la funció. El color el podem modificar de la següent manera:

  Canvas->Brush->Color=COLOR

  On es pot introduir el color desitjat tan sols canviant la constant COLOR, pel nom del color, per exemple si es vol que siga verd, haurem de introduir la constant clGreen:

  Canvas->Brush->Color=clGreen

  De la mateixa manera, també podrem introduir: clRed, clYellow, clBtnFace, clBlack, etc. Segons el color desitjat.

- **El color de les arestes del rectangle:** es pot modificar també aquest valor al que desitjat, per realitzar-ho, actuem de la mateixa forma que abans, però ara utilitzant la següent sentència:

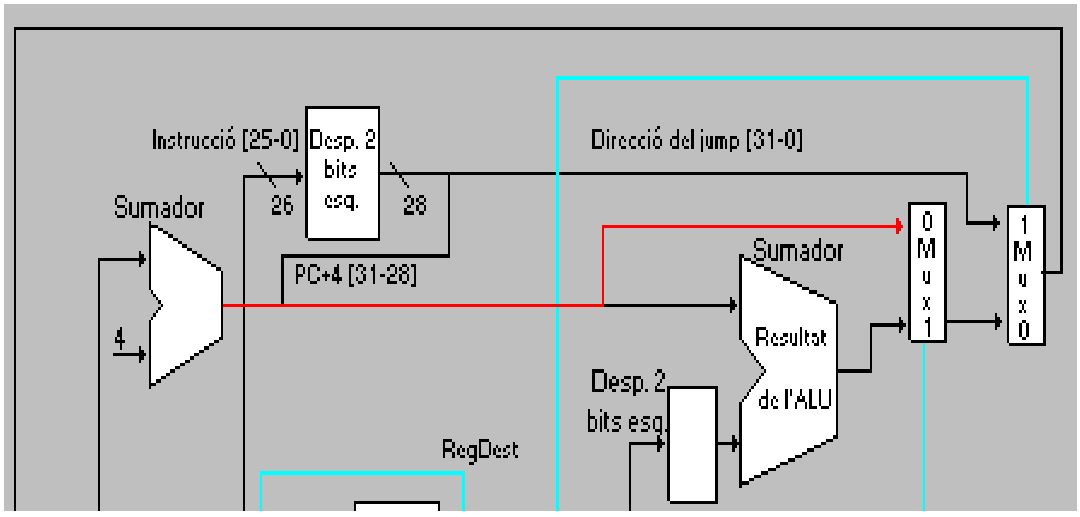
  Canvas->Pen->Color = COLOR;

  Els gamma de colors a introduir són els mateixos que per al fons.

D'altra banda, es troba el tema de les línies que uneixen aquests blocs funcionals de l'arquitectura, aquest ha sigut un aspecte prou complicat, degut principalment a que aquestes línies deuen canviar de color segons s'activen amb els passos de la instrucció. Es per aquest canvi de color de les línies que hem de tindre a algun lloc la informació de totes les línies que trobem al gràfic de la figura 4.1. Aquestes variables són les següents:

POINT LMux5_PC[811],LPC_MemIns[16],LPC_ALUPC[236],L4[21];
POINT LMemIns_Desp1[286],LMemIns_Control[136];
POINT LMemIns_RegLec2[116],LMemIns_Mux1_0[91];
POINT LMemIns_ExtSigne[181],LMemIns_ControlALU[306];
POINT LMux1[26],LMux3_Registres[516],LALUPC_ALUBot[221];
POINT LALUPC_Mux4[316],LALUBot_Mux4[51],LMux4_Mux5[31];
POINT LDesp1_Mux5[414],LDesp2_ALUBot[16],LExtSigne_Desp2[274];
POINT LDadLleg1_ALU[56],LDadLleg2_Mux2[31];
POINT LExtSigne_Mux2[71],LMux2_ALU[16],LALU_MemDad[36];
POINT LALU_Mux3[196],LMemDad_Mux3[26];
POINT LCEscrReg[45],LCFuenteALU[163],LCEscrMem[275];
POINT LCMemaReg[368],LCLeerMem[575],LCSaltoCond[185];
POINT LCSaltoIncond[435],LCRegDest[371],LCCero[120];
POINT LCControlALU_ALU[68];
POINT LMemIns_Mux1_1[81],LMemIns_RegLec1[136];
POINT LDadLleg2_MemDad[171], ,LCALUOp[325];
POINT LCAND_Mux4[82];

Com es veu, totes aquestes variables son vectors de tipus POINT, ja comentat abans, és a dir, per exemple la línia LALUPC_Mux4[316] és un vector de punts que contenen les coordenades de cada punt dins del Canvas, que com es pot suposar es la línia que uneix els blocs de l'ALU que incrementa el PC, amb el multiplexor que es troba a l'extrem superior dret, concretament el que tria entre donar-li eixida a la dada PC+4 ó la dada immediata ja tractada. Ho podem veure gràficament a la següent figura ressaltada amb color roig:



Per a que aquestes variables arriben a contindre els punts (o millor dit, *píxels*) haurem de inicialitzar-les en temps de programació. Per conseguir açò, haurem de fixar-nos amb un punt d'eixida i un punt de finalització, en el nostre cas el punt d'eixida es troba a la dreta del bloc sumador del PC, fins a l'esquerra

del bloc del multiplexor que connecta amb el PC. El codi emprat per realitzar aquesta línia ha sigut:

```
for (i=0;i<160;i++)
{
        LALUPC_Mux4[i].x=LALUPC_ALUBot[i].x;
        LALUPC_Mux4[i].y=LALUPC_ALUBot[i].y;
}
fixat=LALUPC_Mux4[159].y;
for (i=160;i<185;i++)
{
        LALUPC_Mux4[i].x=LALUPC_Mux4[159].x;
        LALUPC_Mux4[i].y=fixat--;
}
fixat=LALUPC_Mux4[184].x;
for (i=185;i<310;i++)

        LALUPC_Mux4[i].x=fixat++;
        LALUPC_Mux4[i].y=LALUPC_Mux4[184].y;
}
LALUPC_Mux4[310].x=LALUPC_Mux4[311].x=LALUPC_Mux4[309].x-1;
LALUPC_Mux4[310].y=LALUPC_Mux4[309].y+1;
LALUPC_Mux4[311].y=LALUPC_Mux4[309].y-1;
LALUPC_Mux4[312].x=LALUPC_Mux4[313].x=LALUPC_Mux4[309].x-2;
LALUPC_Mux4[312].y=LALUPC_Mux4[309].y+1;
LALUPC_Mux4[313].y=LALUPC_Mux4[309].y-1;
LALUPC_Mux4[314].x=LALUPC_Mux4[315].x=LALUPC_Mux4[309].x-2;
LALUPC_Mux4[314].y=LALUPC_Mux4[309].y+2;
LALUPC_Mux4[315].y=LALUPC_Mux4[309].y-2;

for (i=0;i<316;i++)
        Canvas->Pixels[LALUPC_Mux4[i].x][LALUPC_Mux4[i].y]=clBlack;
```

El primer bucle *for* el que fa es igualar el començament de la línia que estem realitzant amb una anterior, així ens podem aprofitar d'aquesta línia ja feta per fer-ne l'actual.

El següent bucle *for* ve precedit de l'assignació de la variable *fixat*, que com el propi nom ho indica es un valor que es queda fixat per poder incrementar-se de manera senzilla per poder dibuixar la línia, açò és, si per exemple volem que la línia es desplace en l'eix vertical, com en l'exemple de baix a dalt, haurem de deixar constant la base del punt on es trobem (eix horitzontal, variable x de la estructura POINT), que com es veu a l'exemple es correspon amb la component 184 i incrementar mitjançant el bucle for la variable y píxel a píxel. També es

veu clarament el funcionament de la variable fixat, on decrementem unitat a unitat el valor per poder dibuixar la línia cap amunt.

El següent bucle for realitza la mateixa funció que l'explicat al paràgraf anterior, sols que ara ens movem per l'eix horitzontal x i com es veu, deixem constant el valor de l'altura.

La següent estructura del programa, que com s'observa és una simple assignació de valors de punts, es correspon al dibuix d'una espècie de fletxa, que com es veu a la figura anterior cadascuna de les línies duu al seu final. Tota aquesta informació es quedarà guardada al vector de cada línia, es recorda que aquest era *LALUPC_Mux4*.

Per últim realitzem un bucle for per dibuixar píxel a píxel la línia elaborada, amb el color negre.

Ara aquesta línia ja la podrem tornar a coloretjar del color que es desitje tornant a passar-li un bucle for idèntic a l'anterior però canviant el color pel preferit.

4.3. Acceptació de la instrucció introduïda.

Al introduir la instrucció des de el teclat, la instrucció ha d'existir a la base de dades del programa, per a que la instrucció no siga errònia ni els registres siguen distints als suportats per l'arquitectura s'han realitzat al cos del programa dues funcions que es passen a explicar.

La primera d'aquestes funcions és l'encarregada de verificar que la instrucció siga correcta, la funció conté el següent codi:

```
int TForm1::BuscaInstruccio (char *cadena)
{
   ///Primer mirem si la instrucció és de tipus R.
   if (strcmp (cadena,"sll") == 0) return 65;
   if (strcmp (cadena,"srl") == 0) return 65;
   if (strcmp (cadena,"sra") == 0) return 65;
   if (strcmp (cadena,"sllv") == 0) return 65;
   if (strcmp (cadena,"srlv") == 0) return 65;
   if (strcmp (cadena,"srav") == 0) return 65;
   if (strcmp (cadena,"jr") == 0) return 65;
   if (strcmp (cadena,"jalr") == 0) return 65;
   if (strcmp (cadena,"syscall") == 0) return 65;
   if (strcmp (cadena,"break") == 0) return 65;
   if (strcmp (cadena,"mfhi") == 0) return 65;
   if (strcmp (cadena,"mthi") == 0) return 65;
```

```
if (strcmp (cadena,"mflo") == 0) return 65;
if (strcmp (cadena,"mtlo") == 0) return 65;
if (strcmp (cadena,"mult") == 0)
{
      tipus_parametre[0]=1;
      tipus_parametre[1]=1;
      funct = 24;
      param_exactes=2;
      return 0;
}
if (strcmp (cadena,"multu") == 0)
{
      tipus_parametre[0]=1;
      tipus_parametre[1]=1;
      funct = 25;
      param_exactes=2;
      return 0;
}
if (strcmp (cadena,"div") == 0)
{
      tipus_parametre[0]=1;
      tipus_parametre[1]=1;
      funct = 26;
      param_exactes=2;
      return 0;
}
if (strcmp (cadena,"divu") == 0)
{
      tipus_parametre[0]=1;
      tipus_parametre[1]=1;
      funct = 27;
      param_exactes=2;
      return 0;
}
if (strcmp (cadena,"add") == 0)
{
      tipus_parametre[0]=1;
      tipus_parametre[1]=1;
      tipus_parametre[2]=1;
      funct = 32;
      param_exactes=3;
      return 0;
}
if (strcmp (cadena,"addu") == 0)
{
      tipus_u=true;
```

```
        tipus_parametre[0]=1;
        tipus_parametre[1]=1;
        tipus_parametre[2]=1;
        funct = 33;
        param_exactes=3;
        return 0;
    }
    if (strcmp (cadena,"sub") == 0)
    {
        tipus_parametre[0]=1;
        tipus_parametre[1]=1;
        tipus_parametre[2]=1;
        funct = 34;
        param_exactes=3;
        return 0;
    }
    if (strcmp (cadena,"subu") == 0)
    {
        tipus_parametre[0]=1;
        tipus_parametre[1]=1;
        tipus_parametre[2]=1;
        funct = 35;
        param_exactes=3;
        return 0;
    }
    if (strcmp (cadena,"and") == 0) return 65;
    if (strcmp (cadena,"or") == 0) return 65;
    if (strcmp (cadena,"xor") == 0) return 65;
    if (strcmp (cadena,"nor") == 0) return 65;
    if (strcmp (cadena,"slt") == 0)
    {
        tipus_parametre[0]=1;
        tipus_parametre[1]=1;
        tipus_parametre[2]=1;
        funct=42;
        param_exactes=3;
        return 0;
    }
    if (strcmp (cadena,"sltu") == 0)
    {
        tipus_parametre[0]=1;
        tipus_parametre[1]=1;
        tipus_parametre[2]=1;
        funct=43;
        param_exactes=3;
        return 0;
```

```
}

///Ara els demés cassos.
      if (strcmp (cadena,"j") == 0)
{
      tipus_parametre[0]=2;
      param_exactes = 1;
      return 2;
}
if (strcmp (cadena,"jl") == 0) return 65;
if (strcmp (cadena,"beq") == 0)
{
      tipus_parametre[0]=1;
      tipus_parametre[1]=1;
      tipus_parametre[2]=2;
      param_exactes = 3;
      return 4;
}
if (strcmp (cadena,"bne") == 0)
{
      tipus_parametre[0]=1;
      tipus_parametre[1]=1;
      tipus_parametre[2]=2;
      param_exactes=3;
      return 5;
}
if (strcmp (cadena,"blez") == 0) return 65;
if (strcmp (cadena,"bgtz") == 0) return 65;
if (strcmp (cadena,"addi") == 0)
{
      tipus_parametre[0]=1;
      tipus_parametre[1]=1;
      tipus_parametre[2]=2;
      param_exactes=3;
      return 8;
}
if (strcmp (cadena,"addiu") == 0)
{
      tipus_parametre[0]=1;
      tipus_parametre[1]=1;
      tipus_parametre[2]=2;
      param_exactes=3;
      return 9;
}
if (strcmp (cadena,"slti") == 0) return 65; ///10
if (strcmp (cadena,"situ") == 0) return 65;
```

```
    if (strcmp (cadena,"andi") == 0) return 65;
    if (strcmp (cadena,"ori") == 0) return 65;
    if (strcmp (cadena,"xori") == 0) return 65;
    if (strcmp (cadena,"lui") == 0) return 65;
    if (strcmp (cadena,"lb") == 0) return 65;
    if (strcmp (cadena,"lh") == 0) return 65;
    if (strcmp (cadena,"lwl") == 0) return 65;
    if (strcmp (cadena,"lw") == 0)
    {
        tipus_parametre[0]=1;
        tipus_parametre[1]=2;
        tipus_parametre[2]=1;
        param_exactes=3;
        return 35;
    }
    if (strcmp (cadena,"lbu") == 0) return 65;
    if (strcmp (cadena,"lhu") == 0) return 65;
    if (strcmp (cadena,"lwr") == 0) return 65;
    if (strcmp (cadena,"sb") == 0) return 65;
    if (strcmp (cadena,"sh") == 0) return 65;
    if (strcmp (cadena,"swl") == 0) return 65;
    if (strcmp (cadena,"sw") == 0)
    {
        tipus_parametre[0]=1;
        tipus_parametre[1]=2;
        tipus_parametre[2]=1;
        param_exactes=3;
        return 43;
    }
    if (strcmp (cadena,"swr") == 0) return 65;
    if (strcmp (cadena,"lwc0") == 0) return 65;
    if (strcmp (cadena,"lwc1") == 0) return 65;
    if (strcmp (cadena,"swc0") == 0) return 65;
    if (strcmp (cadena,"swc1") == 0) return 65;

    ///Es tornarà el valor 64 si la instrucció no existeix.
    return (64);
}
```

Cal comentar per poder explicar aquesta funció, que abans de cridar-la s'ha extret de la instrucció introduïda la operació a executar, prèvia pulsació del botó *Acceptar*, d'ací que una vegada entrem a la funció anem comparant amb totes les instruccions possibles a l'arquitectura MIPS. Per tant, una vegada es troba la operació, és a dir, la comparació fa que s'introduisca dins d'un *if*, s'inicialitcen unes certes variables, que són:

- *tipus_parametre: É*s un vector de tres components, el qual emmagatzemarà el tipus de contingut de cadascun dels paràmetres de la instrucció. D'aquesta manera 1 correspon a registre i 2 correspon a nombre, així ens evitarem un ús incorrecte en la manera d'introduir la instrucció.

- *param_exacte*: Aquesta variable guardarà el nombre de paràmetres que han d'acompanyar a l'operació. Aquesta variable i l'anterior com es pot arribar a pensar ens serviran per a la funció BuscaCamps explicada més avant.

- Per últim la funció tornarà un valor que es correspon, en cas que la funció estiga definida, amb el codi d'operació de la instrucció comentat al capítol 2. Pel contrari si la instrucció no ha estat definida tornarà el valor 65, mentre que si l'operació no existeix al repertori de instruccions de l'arquitectura, tornarà el valor 64.

La segona de les funcions per a l'acceptació de la instrucció introduïda, és la següent

```
bool TForm1::BuscaCamps (void)
{
        int punter, j=0, k; ///k l'inicialitzem després per estalviar.
        char param_correctes[33][10]={"$zero","~","$v0","$v1","$a0","$a1",
        "$a2","$a3","$t0","$t1","$t2","$t3","$t4","$t5","$t6","$t7",
        "$s0","$s1","$s2","$s3","$s4","$s5","$s6","$s7",
        "$t8","$t9","~","~","$gp","$sp","$fp","$ra"};
        bool param_valid=0;

   ///Copiem el nom de la constant o immediat de la finestra dels registres
   ///en la posició corresponent.
   strcpy (param_correctes[32],nom_immediat);
   ///Comencem a observar el nemònic a partir d'on hem quedat abans,
   ///després d'haver llegit la instrucció.
   punter = strlen(Instruccio)+1;

   ///Ara trobarem tots els paràmetres de la instrucció.
   while (Nemonic[punter] != '\0')
   {
        k=0;
        while (Nemonic[punter] != ',' && Nemonic[punter] != '\0' &&
        Nemonic[punter] != '(' && Nemonic[punter] != ')')
      {
        parametre[j][k] = Nemonic[punter];
        punter++;
```

```
        k++;
      }
      punter++; /// Si no, no bota la coma.
      j++;
      if (j > 3) return 0; /// Si hi ha més paràmetres que els que toca.
    }
    if (j==0) return 0; ///Si no hi ha paràmetres també s'en ix.
    if (param_exactes != j) return 0; /// Si no hi ha els params. exactes.
    nombre_parametres = j;

    j=-1; ///D'aquesta manera podrem comparar després si el paràmetre
          ///és correcte o no.

    /// Ara compararem els paràmetres que tenim a la instrucció amb
    ///els paràmetres vàlids de la taula anterior.
    /// Aquest for no detectarà una posició incorrecta de l'immediat,
    ///degut a què segons la instrucció aquest es situa en diferent lloc.
    for (k=0;k<nombre_parametres;k++)
    {
        while (!param_valid && j<33) ///param_valid s'ha inicialitzat a 0.
      {
        j++;
        if (strcmp (parametre[k],param_correctes[j]) == 0)
        {
           registre[k]=j;///Si és el NOM de l'immediat, j=33.
           param_valid=1;
           j=-1;

           if (tipus_parametre[k] == 2) return 0; ///Deuria ser 1:registre
        }
      }
      if (!param_valid) ///Provarem transformant a número a veure
                        ///si el paràmetre és una constant.
      {
        if (!Es_Numero (parametre[k])) return 0; ///No és un nombre.

         param_valid=0; ///Tant si és el nom de l'immediat com un valor
         j=-1;          ///numèric fem açò.

         if (tipus_parametre[k] == 1) return 0; ///Deuria ser 2.
      }
      else param_valid=0;
    }
    return 1; ///Si tot ha anat bé tornem un 1.
}
```

Com que l'explicació de la funció mitjançant els comentaris introduïts pot no ser comprensible per al lector, es passa a explicar les diferents parts que la formen.

Primer declarem tres variables per a utilitzar a la funció, la més important d'aquestes potser siga *param_correctes* que justament són els possibles registres que suporta l'arquitectura, que com veure'm després, serviran per a comparar després amb els camps introduïts.

A continuació, com que ja dúiem d'abans llegida l'operació, el que fem ara és anar llegint des de el punt on s'hem trobat fins al final de la cadena que conté la instrucció, per tant el que busquem ara són els registres o valors numèrics introduïts a la instrucció. El que fem primer és extraure tots els camps de la funció i guardar-los al vector *paràmetres* com a cadenes de caràcters.

Ara és el torn per a que s'examine cadascun dels components del vector parametres, aquest bucle s'encarrega per una banda de comprovar si el paràmetre és un dels registres inicialitzats abans o en canvi un nombre. Per a comprovar si el paràmetre es un valor numèric, es fa crida a la funció *EsNumero*, que comprova si certament ho és. En cas que el paràmetre que s'analitza a cadascuna de les iteracions de búsqueda de paràmetres determina que el paràmetre no és un registre o un valor, la funció tornarà el valor 0 (realment *false* degut a que la funció s'ha declarat com a *bool*, açò és, ha de tornar un valor booleà), mentre que si després d'examinar tots els paràmetres no dóna cap error, tornarà 1 (o *true*).

4.4. Canviar el color de les línies pas a pas.

Una vegada dibuixada l'arquitectura i realitzada la comprovació que la instrucció ha sigut introduïda correctament, passem a la simulació de l'arquitectura.

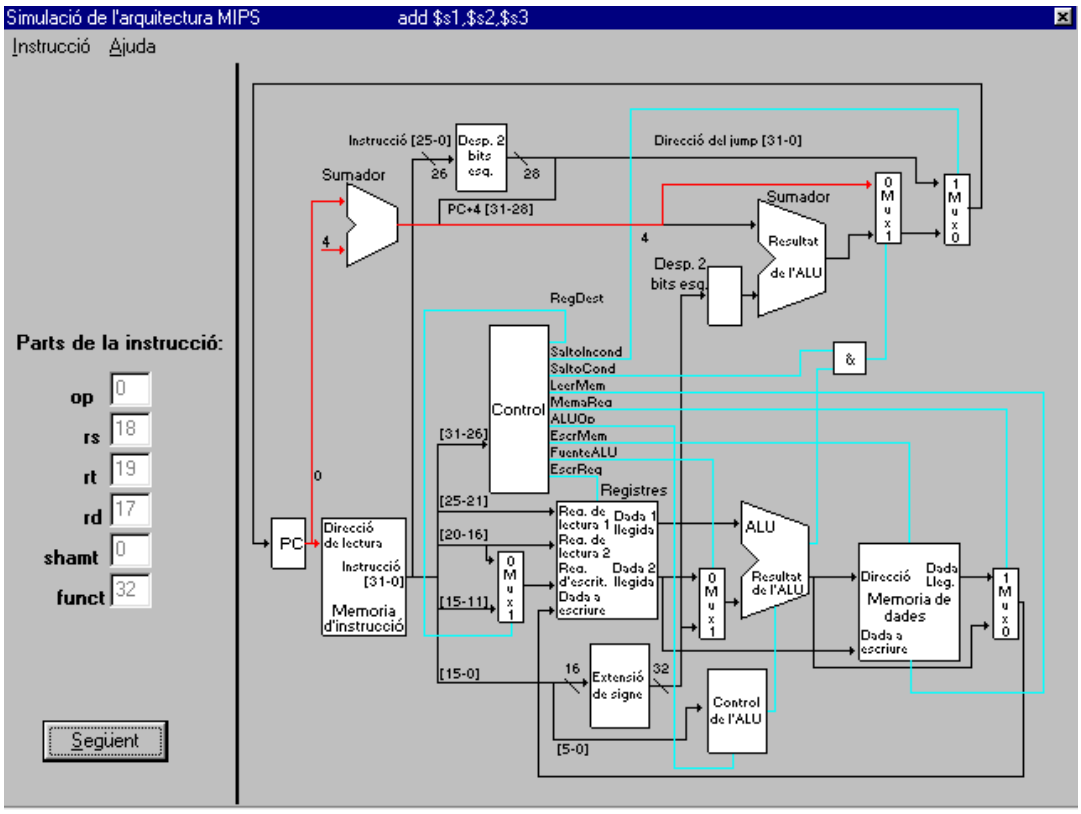
Per a realitzar aquesta simulació, s'ha fet ús d'un objecte *TButton* comentat al capítol 3, que ens serveix per poder executar pas a pas la instrucció. Aquest botó duu el text Següent, indicant així que al polsar-lo passem a la següent instrucció.

Al polsar per primera vegada a Següent, el que fem és activar una sèrie de Llindes que són comunes per a totes les instruccions i que per tant s'ha realitzat mitjançant una nova funció anomenada *Pas1*, el codi de la qual es mostrat tot seguit:

```
void TForm1::Pas1 (void)
{
        int i;
        char c[15];
```

```
        //Aquest pas és comú per totes les linees.
        for (i=0;i<16;i++)
                Canvas->Pixels[LPC_MemIns[i].x][LPC_MemIns[i].y] = clRed;
        for (i=0;i<236;i++)
                Canvas->Pixels[LPC_ALUPC[i].x][LPC_ALUPC[i].y] = clRed;
        for (i=0;i<21;i++)
                Canvas->Pixels[L4[i].x][L4[i].,y] = clRed;
        for (i=0;i<316;i++)
             Canvas->Pixels[LALUPC_Mux4[i].x][LALUPC_Mux4[i].y]= clRed;

        Canvas->Brush->Color = clBtnFace;
        itoa(rpc,c,10);
        Canvas->TextRect(Contingut[1],Contingut[1].Left,Contingut[1].Top,c);
        itoa(rpc+4,c,10);
        Canvas->TextRect(Contingut[2],Contingut[2].Left,Contingut[2].Top,c);
}
```

L'únic que es realitza en aquesta funció, és el canvi de color de les línies tal i com abans s'ha explicat i posteriorment la introducció mitjançant text del contingut de dues de les línies, el funcionament es pot veure gràficament a la següent figura:



A partir d'ara el canvi de color de les línies serà igual per a totes les instruccions, el que s'ha pensat en a l'hora de la programació és el següent:

- Com que tenim que tindre en compte la existència de múltiples instruccions, a més de la existència d'uns passos independents per a cada instruccions tant en nombre com en activació de les línies, la solució pressa ha estat la creació d'una variable tridimensional:

  char Linees[64][5][32];

  Cadascuna d'aquestes dimensions té un significat, és a dir, la primera dimensió (64), indica que es va a reservar espai per un conjunt de 64 instruccions (encara que com sabem, no les aprofitarem totes). La segona dimensió especifica el nombre de passos màxim que podem introduir a cadascuna d'aquestes 64 instruccions. Per últim, la tercera dimensió es correspon amb el nombre total de línies existents a l'arquitectura.

- Ara, introduirem a aquesta matriu tridimensional de caràcters el contingut de les línies a activar a cada pas. El criteri seguit per a la inicialització de cada component de la matriu, ha sigut: quan el caràcter siga '1', la línia, de les 32 possibles s'activarà, mentre que si és '0', no canviarà el seu color. Podem veure al tros de codi següent:

  ```
  strcpy (Linees[0][0],"10000111000100000100011100100000");
  strcpy (Linees[0][1],"10000111000100000100011100101000");
  strcpy (Linees[0][2],"10000111010111000100011100101000");
  strcpy (Linees[0][3],"x");
  ```

  Com s'ha explicat abans, les sentències el que realitzen es inicialitzar les cadenes de caràcters de les instruccions de tipus R (codi d'operació 0). Vegem com la instrucció va a tindre 4 passos (s'ha d'afegir el realitzat a la funció Pas1) per a cada pas es fàcil preveure que s'activaran les línies marcades amb el caràcter '1'.

- Per últim, per a que les línies canvien de color, simulant així que s'activen, cridem a la funció *Activar*, el codi de la qual es mostra a continuació:

```
void TForm1::Activar (void)
{
        int i,j;
```

```
        j=codi_operacio;

        //Primer mirem les linees de control a activar.
        if (LControl[j][0] == '1')
        {
                for (i=0;i<371;i++)
                Canvas->Pixels[LCRegDest[i].x][LCRegDest[i].y] = clBlue;
        }

        if (LControl[j][1] == '1')
        {
                for (i=0;i<163;i++)
                Canvas->Pixels[LCFuenteALU[i].x][LCFuenteALU[i].y]          =
clBlue;
        }

        if (LControl[j][2] == '1')
        {
                for (i=0;i<368;i++)
                Canvas->Pixels[LCMemaReg[i].x][LCMemaReg[i].y] = clBlue;
        }

        if (LControl[j][3] == '1')
        {
                for (i=0;i<45;i++)
                Canvas->Pixels[LCEscrReg[i].x][LCEscrReg[i].y] = clBlue;
        }

        if (LControl[j][4] == '1')
        {
                for (i=0;i<575;i++)
                Canvas->Pixels[LCLeerMem[i].x][LCLeerMem[i].y] = clBlue;
        }

        if (LControl[j][5] == '1')
        {
                for (i=0;i<275;i++)
                Canvas->Pixels[LCEscrMem[i].x][LCEscrMem[i].y] = clBlue;
        }

        if (LControl[j][6] == '1')
        {
                for (i=0;i<185;i++)
                Canvas->Pixels[LCSaltoCond[i].x][LCSaltoCond[i].y] = clBlue;
        }
```

```
        if (codi_operacio != 2)
        {
                for (i=0;i<325;i++)
                 Canvas->Pixels[LCALUOp[i].x][LCALUOp[i].y] = clBlue;

    Canvas->TextRect(Contingut[7],Contingut[7].Left,Contingut[7].Top,eALU);
}
        if (LControl[j][9] == '1')
        {
                for (i=0;i<435;i++)
                 Canvas->Pixels[LCSaltoIncond[i].x][LCSaltoIncond[i].y]          =
clBlue;
        }

        if (LControl[j][10] == '1')
        {
                for (i=0;i<68;i++)
        Canvas->Pixels[LCControlALU_ALU[i].x][LCControlALU_ALU[i].y]  =
        clBlue;

   Canvas->TextRect(Contingut[8],Contingut[8].Left,Contingut[8].Top,ALUc);
  }

  // Ara les linees "normals".

       if (Linees[j][pas-1][0] == '1')
      {
        for (i=0;i<811;i++)
        Canvas->Pixels[LMux5_PC[i].x][LMux5_PC[i].y] = clRed;

        Canvas->TextRect(Contingut[0],Contingut[0].Left,Contingut[0].Top,
        String(valor[0]));
      }

      if (Linees[j][pas-1][1] == '1')
      {
                for (i=0;i<16;i++)
                 Canvas->Pixels[LPC_MemIns[i].x][LPC_MemIns[i].y] = clRed;

        Canvas->TextRect(Contingut[1],Contingut[1].Left,Contingut[1].Top,
        String(valor[1]));
      }

      if (Linees[j][pas-1][2] == '1')
      {
```

```
                for (i=0;i<236;i++)
                 Canvas->Pixels[LPC_ALUPC[i].x][LPC_ALUPC[i].y] = clRed;

      Canvas->TextRect(Contingut[1],Contingut[1].Left,Contingut[1].Top,
      String(valor[1]));
    }

    if (Linees[j][pas-1][3] == '1')
    {
                for (i=0;i<21;i++)
                 Canvas->Pixels[L4[i].x][L4[i].y] = clRed;
    }

    if (Linees[j][pas-1][4] == '1')
    {
       for (i=0;i<286;i++)
       Canvas->Pixels[LMemIns_Desp1[i].x][LMemIns_Desp1[i].y] = clRed;
    }

     if (Linees[j][pas-1][5] == '1')
    {
       for (i=0;i<136;i++)
       Canvas->Pixels[LMemIns_Control[i].x][LMemIns_Control[i].y]      =
clRed;
    }

    if (Linees[j][pas-1][6] == '1')
    {
      for (i=0;i<136;i++)
    Canvas->Pixels[LMemIns_RegLec1[i].x][LMemIns_RegLec1[i].y] = clRed;
    }

    if (Linees[j][pas-1][7] == '1')
    {
      for (i=0;i<116;i++)
    Canvas->Pixels[LMemIns_RegLec2[i].x][LMemIns_RegLec2[i].y] = clRed;
    }

    if (Linees[j][pas-1][8] == '1')
    {
      for (i=0;i<91;i++)
     Canvas->Pixels[LMemIns_Mux1_0[i].x][LMemIns_Mux1_0[i].y] = clRed;
    }

    if (Linees[j][pas-1][9] == '1')
    {
```

```
  for (i=0;i<81;i++)
 Canvas->Pixels[LMemIns_Mux1_1[i].x][LMemIns_Mux1_1[i].y] = clRed;
}

if (Linees[j][pas-1][10] == '1')
{
  for (i=0;i<181;i++)
Canvas->Pixels[LMemIns_ExtSigne[i].x][LMemIns_ExtSigne[i].y] = clRed;
}

if (Linees[j][pas-1][11] == '1')
{
  for (i=0;i<306;i++)
  Canvas->Pixels[LMemIns_ControlALU[i].x]
  [LMemIns_ControlALU[i].y]=clRed;
}

if (Linees[j][pas-1][12] == '1')
{
  for (i=0;i<26;i++)
  Canvas->Pixels[LMux1[i].x][LMux1[i].y] = clRed;
}

if (Linees[j][pas-1][13] == '1')
{
  for (i=0;i<516;i++)
  Canvas->Pixels[LMux3_Registres[i].x][LMux3_Registres[i].y] = clRed;
}

if (Linees[j][pas-1][14] == '1')
{
  for (i=0;i<221;i++)
Canvas->Pixels[LALUPC_ALUBot[i].x][LALUPC_ALUBot[i].y] = clRed;

  Canvas->TextRect(Contingut[2],Contingut[2].Left,Contingut[2].Top,
  String(valor[2]));
}

if (Linees[j][pas-1][15] == '1')
{
  for (i=0;i<316;i++)
  Canvas->Pixels[LALUPC_Mux4[i].x][LALUPC_Mux4[i].y] = clRed;

  Canvas->TextRect(Contingut[2],Contingut[2].Left,Contingut[2].Top,
  String(valor[2]));
}
```

```
if (Linees[j][pas-1][16] == '1')
{
  for (i=0;i<51;i++)
  Canvas->Pixels[LALUBot_Mux4[i].x][LALUBot_Mux4[i].y] = clRed;

  Canvas->TextRect(Contingut[10],Contingut[10].Left,Contingut[10].Top,
  String(valor[10]));
}

if (Linees[j][pas-1][17] == '1')
{
  for (i=0;i<31;i++)
  Canvas->Pixels[LMux4_Mux5[i].x][LMux4_Mux5[i].y] = clRed;
}

if (Linees[j][pas-1][18] == '1')
{
  for (i=0;i<414;i++)
  Canvas->Pixels[LDesp1_Mux5[i].x][LDesp1_Mux5[i].y] = clRed;

  Canvas->TextRect(Contingut[3],Contingut[3].Left,Contingut[3].Top,
  String(valor[3]));
}

if (Linees[j][pas-1][19] == '1')
{
  for (i=0;i<31;i++)
  Canvas->Pixels[LDesp2_ALUBot[i].x][LDesp2_ALUBot[i].y] = clRed;

  Canvas->TextRect(Contingut[12],Contingut[12].Left,Contingut[12].Top,
  String(valor[12]));
}

if (Linees[j][pas-1][20] == '1')
{
  for (i=0;i<274;i++)
  Canvas->Pixels[LExtSigne_Desp2[i].x][LExtSigne_Desp2[i].y] = clRed;

  Canvas->TextRect(Contingut[4],Contingut[4].Left,Contingut[4].Top,
  String(valor[4]));
}

if (Linees[j][pas-1][21] == '1')
{
  for (i=0;i<56;i++)
```

```
    Canvas->Pixels[LDadLleg1_ALU[i].x][LDadLleg1_ALU[i].y] = clRed;

    Canvas->TextRect(Contingut[5],Contingut[5].Left,Contingut[5].Top,
    String(valor[5]));
  }

  if (Linees[j][pas-1][22] == '1')
  {
    for (i=0;i<31;i++)
    Canvas->Pixels[LDadLleg2_Mux2[i].x][LDadLleg2_Mux2[i].y] = clRed;

    Canvas->TextRect(Contingut[6],Contingut[6].Left,Contingut[6].Top,
    String(valor[6]));
  }

  if (Linees[j][pas-1][23] == '1')
  {
    for (i=0;i<171;i++)
    Canvas->Pixels[LDadLleg2_MemDad[i].x]
    [LDadLleg2_MemDad[i].y] = clRed;

    Canvas->TextRect(Contingut[6],Contingut[6].Left,Contingut[6].Top,
    String(valor[6]));
  }

  if (Linees[j][pas-1][24] == '1')
  {
    for (i=0;i<71;i++)
    Canvas->Pixels[LExtSigne_Mux2[i].x][LExtSigne_Mux2[i].y] = clRed;

    Canvas->TextRect(Contingut[4],Contingut[4].Left,Contingut[4].Top,
    String(valor[4]));
  }

  if (Linees[j][pas-1][25] == '1')
  {
    for (i=0;i<16;i++)
    Canvas->Pixels[LMux2_ALU[i].x][LMux2_ALU[i].y] = clRed;
  }

  if (Linees[j][pas-1][26] == '1')
  {
    for (i=0;i<36;i++)
    Canvas->Pixels[LALU_MemDad[i].x][LALU_MemDad[i].y] = clRed;

    Canvas->TextRect(Contingut[9],Contingut[9].Left,Contingut[9].Top,
```

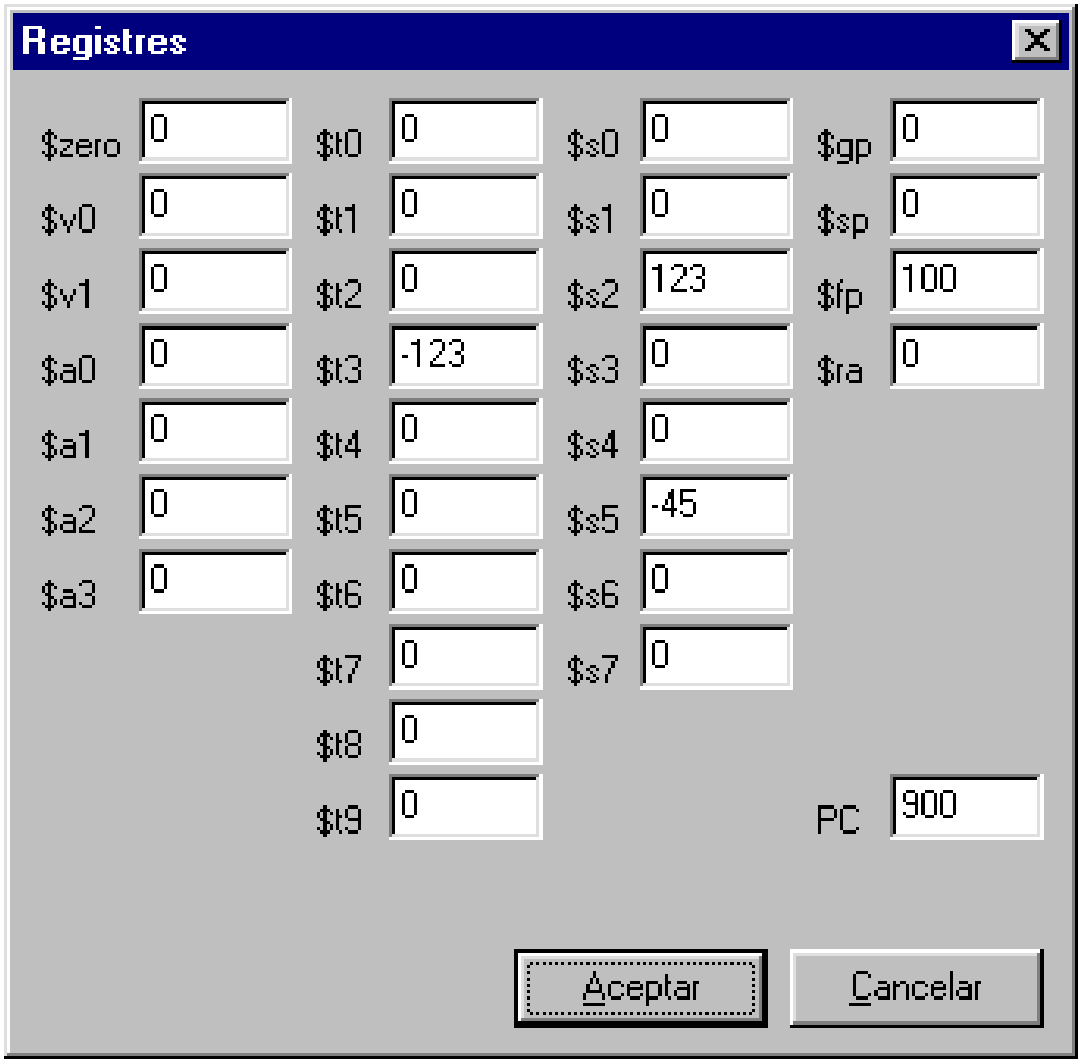
```
      String(valor[9]));
    }

    if (Linees[j][pas-1][27] == '1')
    {
      for (i=0;i<196;i++)
      Canvas->Pixels[LALU_Mux3[i].x][LALU_Mux3[i].y] = clRed;

      Canvas->TextRect(Contingut[9],Contingut[9].Left,Contingut[9].Top,
      String(valor[9]));
    }

    if (Linees[j][pas-1][28] == '1')
    {
      for (i=0;i<26;i++)
      Canvas->Pixels[LMemDad_Mux3[i].x][LMemDad_Mux3[i].y] = clRed;

      Canvas->TextRect(Contingut[11],Contingut[11].Left,Contingut[11].Top,
      String(valor[11]));
    }

    //També situem ací les línies del Cero i de AND_ALUBot
    if (Linees[j][pas-1][29] == '1')
    {
      for (i=0;i<120;i++)
      Canvas->Pixels[LCCero[i].x][LCCero[i].y] = clBlue;
    }

    if (Linees[j][pas-1][30] == '1')
    {
      for (i=0;i<82;i++)
      Canvas->Pixels[LCAND_Mux4[i].x][LCAND_Mux4[i].y] = clBlue;
    }
}
```

Tal com es veu al codi anterior, el que fem és repassar el caràcter corresponent a cada línia i en cas que siga '1', la línia canviarà a color roig, indicant així l'activació de la línia. A més, si la línia que s'activa conté quadres de text, aquest també s'activarà indicant el valor de la línia a cada moment.

4.5. Finestres afegides a la principal.

Al projecte, a més de la finestra principal on es realitza la major part de l'execució del programa, s'han introduït dues finestres més per donar-li una simulació més vertadera. Aquestes dues finestres, com ja sabem són dues noves classes amb els seus objectes individuals.

La primera finestra a comentar és la dels registres, la qual té la següent aparença:



Com es pot observar, aquesta finestra serveix per a introduir valors als registres i al comptador de programa (PC). A més afegeix la possibilitat de visualitzar la realització de la instrucció.

Realment hi ha registres com ara *$gp*, *$sp*, *$fp* i *$ra* que no realitzen realment la seva funció, degut a que el simulador sols suporta l'execució d'una sola execució cada vegada que s'executa. Així per tant com exemple el registre

$sp es correspon amb el registre de punter de pila (*stack pointer*) descrit al capítol 2, però realment no es modifica, encara que si es poden realitzar operacions amb tots ells.

Com es suposa, l'intercanvi de dades entre la finestra principal i la dels registres es constant, per a que existisca aquest flux existeixen dos funcions, una que quan fem aparèixer la finestra dels registres aparega amb els valors de cada registre, la qual és la següent:

```
void __fastcall TForm1::Registres1Click(TObject *Sender)
{
        unsigned r_u[32];
        int q;
        char cadena[15];///El màxim unsigned té 10 dígits 2^32.

        if (!tipus_u)
        {
                Reg->E2->Text = AnsiString (r[2]);
                Reg->E3->Text = AnsiString (r[3]);
                Reg->E4->Text = AnsiString (r[4]);
                Reg->E5->Text = AnsiString (r[5]);
                Reg->E6->Text = AnsiString (r[6]);
                Reg->E7->Text = AnsiString (r[7]);
                Reg->E8->Text = AnsiString (r[8]);
                Reg->E9->Text = AnsiString (r[9]);
                Reg->E10->Text = AnsiString (r[10]);
                Reg->E11->Text = AnsiString (r[11]);
                Reg->E12->Text = AnsiString (r[12]);
                Reg->E13->Text = AnsiString (r[13]);
                Reg->E14->Text = AnsiString (r[14]);
                Reg->E15->Text = AnsiString (r[15]);
                Reg->E16->Text = AnsiString (r[16]);
                Reg->E17->Text = AnsiString (r[17]);
                Reg->E18->Text = AnsiString (r[18]);
                Reg->E19->Text = AnsiString (r[19]);
                Reg->E20->Text = AnsiString (r[20]);
                Reg->E21->Text = AnsiString (r[21]);
                Reg->E22->Text = AnsiString (r[22]);
                Reg->E23->Text = AnsiString (r[23]);
                Reg->E24->Text = AnsiString (r[24]);
                Reg->E25->Text = AnsiString (r[25]);
                Reg->E28->Text = AnsiString (r[28]);
                Reg->E29->Text = AnsiString (r[29]);
                Reg->E30->Text = AnsiString (r[30]);
                Reg->E31->Text = AnsiString (r[31]);
```

```
        }
        else
        {

                for (q=0;q<32;q++) r_u[q]=r[q];


                Reg->E2->Text = itoa(r_u[2],cadena,10);
                Reg->E3->Text = itoa(r_u[3],cadena,10);
                Reg->E4->Text = itoa(r_u[4],cadena,10);
                Reg->E5->Text = itoa(r_u[5],cadena,10);
                Reg->E6->Text = itoa(r_u[6],cadena,10);
                Reg->E7->Text = itoa(r_u[7],cadena,10);
                Reg->E8->Text = itoa(r_u[8],cadena,10);
                Reg->E9->Text = itoa(r_u[9],cadena,10);
                Reg->E10->Text = itoa(r_u[10],cadena,10);
                Reg->E11->Text = itoa(r_u[11],cadena,10);
                Reg->E12->Text = itoa(r_u[12],cadena,10);
                Reg->E13->Text = itoa(r_u[13],cadena,10);
                Reg->E14->Text = itoa(r_u[14],cadena,10);
                Reg->E15->Text = itoa(r_u[15],cadena,10);
                Reg->E16->Text = itoa(r_u[16],cadena,10);
                Reg->E17->Text = itoa(r_u[17],cadena,10);
                Reg->E18->Text = itoa(r_u[18],cadena,10);
                Reg->E19->Text = itoa(r_u[19],cadena,10);
                Reg->E20->Text = itoa(r_u[20],cadena,10);
                Reg->E21->Text = itoa(r_u[21],cadena,10);
                Reg->E22->Text = itoa(r_u[22],cadena,10);
                Reg->E23->Text = itoa(r_u[23],cadena,10);
                Reg->E24->Text = itoa(r_u[24],cadena,10);
                Reg->E25->Text = itoa(r_u[25],cadena,10);
                Reg->E28->Text = itoa(r_u[28],cadena,10);
                Reg->E29->Text = itoa(r_u[29],cadena,10);
                Reg->E30->Text = itoa(r_u[30],cadena,10);
                Reg->E31->Text = itoa(r_u[31],cadena,10);
        }
        Reg->EPC->Text = AnsiString (rpc);

        Reg->Show();
}
```

L'únic que es fa, com es veu és introduir el valor dels registres, el qual es troba emmagatzemat a un vector anomenat *r*, als *Edits* de la finestra de registres.

L'altra de les funcions és:

```
void __fastcall TForm2::Button1Click(TObject *Sender)
{
        Form1->r[2]=E2->Text.ToInt();
        Form1->r[3]=E3->Text.ToInt();
        Form1->r[4]=E4->Text.ToInt();
        Form1->r[5]=E5->Text.ToInt();
        Form1->r[6]=E6->Text.ToInt();
        Form1->r[7]=E7->Text.ToInt();
        Form1->r[8]=E8->Text.ToInt();
        Form1->r[9]=E9->Text.ToInt();
        Form1->r[10]=E10->Text.ToInt();
        Form1->r[11]=E11->Text.ToInt();
        Form1->r[12]=E12->Text.ToInt();
        Form1->r[13]=E13->Text.ToInt();
        Form1->r[14]=E14->Text.ToInt();
        Form1->r[15]=E15->Text.ToInt();
        Form1->r[16]=E16->Text.ToInt();
        Form1->r[17]=E17->Text.ToInt();
        Form1->r[18]=E18->Text.ToInt();
        Form1->r[19]=E19->Text.ToInt();
        Form1->r[20]=E20->Text.ToInt();
        Form1->r[21]=E21->Text.ToInt();
        Form1->r[22]=E22->Text.ToInt();
        Form1->r[23]=E23->Text.ToInt();
        Form1->r[24]=E24->Text.ToInt();
        Form1->r[25]=E25->Text.ToInt();
        Form1->r[28]=E28->Text.ToInt();
        Form1->r[29]=E29->Text.ToInt();
        Form1->r[30]=E30->Text.ToInt();
        Form1->r[31]=E31->Text.ToInt();
        Form1->rpc=EPC->Text.ToInt();

        Hide ();
}
```

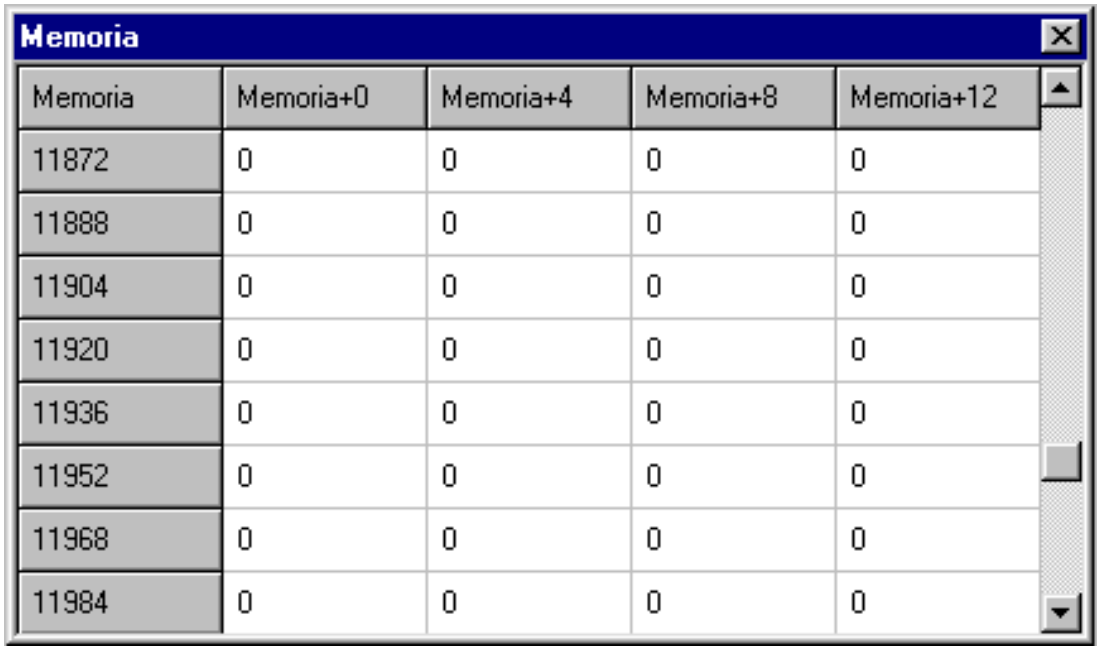
Aquesta nova funció s'encarrega de quan es polse el botó Acceptar de la finestra dels registres, emmagatzemar els canvis introduïts a aquesta finestra, al vector *r*, ja esmentat.

L'altra de les finestres abans esmentades, és la que simula la memòria de dades. A aquesta finestra es podrà com a la anterior finestra de registres, introduir nous valors a les posicions de memòria possibles. Aquestes posicions van des de

la posició 0 fins a la 16000, indicant així que és possible l'emmagatzematge de 16000 paraules o *words*, ja comentades al capítol 1.

També, com és lògic, es possible la visualització de la memòria una vegada efectuada una operació d'accés a memòria, com són *sw* i *lw*, ja explicades al capítol 1.

L'aspecte de la finestra de memòria, és el següent:



La forma de tractar aquesta finestra per part de Borland Builder, és idèntica a que es tractara d'una matriu, en aquest cas es tracta d'una matriu de cinc columnes i mil i una files, encara que com s'observa al veure la figura anterior, aprofitable és una matriu de 4x1000, degut a que una fila i una columna s'aprofita per a indicació de les posicions de memòria.

D'entre les funcions necessàries per fer funcionar correctament aquesta finestra, cal destacar les següents:

La primera és l'encarregada de escriure tot el text que indica les posicions de memòria, a la figura anterior, es tracta del text amb el fons de color gris, el codi és:

```
void __fastcall TForm3::TaulaDrawCell(TObject *Sender, long Col, long Row,
        TRect &Rect, TGridDrawState State)
{
        AnsiString cadenes[4]={"+0","+4","+8","+12"};
```

```
        AnsiString cadena="Memoria";

        if (Col == 0 && Row == 0)
                Taula->Canvas->TextOut(Rect.Left+5,Rect.Top+5,cadena);
        else if (Row == 0)
                Taula->Canvas->TextOut(Rect.Left+5,Rect.Top+5,
                cadena+cadenes[Col-1]);
        else if (Col == 0)
                Taula->Canvas->TextOut(Rect.Left+5,Rect.Top+5,
                AnsiString(val[Row-1]));
        else if (Col != 5)
                Taula->Canvas->TextOut(Rect.Left+5,Rect.Top+5,
                AnsiString(Cont[Col-1][Row-1]));
}
```

Aquesta funció comença com es veu a la columna 0 i fila 0, i escriu el text de *cadena* que com es veu abans, escriu la paraula *Memòria*. També com s'observa a la fila 0, escriurà el text Memòria al que concatena els texts *+0*, *+4*, *+8* i *+12*, indicant així la posició de la memòria que es tracta. De la mateixa manera, per a la primera columna (columna 0), introduirem per a cada component el valor corresponent, que s'extrau del vector de nombres enters *val*, que abans s'ha inicialitzat amb els valors múltiples de 16 corresponents. Per últim, introduirem els valors que indiquen el contingut de la posició de memòria corresponent. Tots aquests valors es guarden al vector *Cont* que no és més que una matriu amb les dimensions 4x1000.

L'anterior funció s'executa cada vegada que obrim la finestra de memòria. Però per a que el contingut de la memòria canvie realment, hem de modificar el valor del vector *Cont* ja esmentat. En canvi a com es podia pensar, no s'ha realitzat cap funció per a modificar aquest vector, el que s'ha fet ha sigut: com que tan sols hi ha al simulador dues funcions que realitzen accessos a memòria, *lw* i *sw*, aleshores durant l'execució d'aquestes instruccions es quan es modifica aquest vector *Cont*, com a exemple vegem el codi efectuat per a la realització de *sw*, que s'encarrega d'emmagatzemar a la memòria un valor:

```
pos=valor[9]%4;
if (pos != 0)
        ShowMessage ("La direcció no és correcta");
else
{
        pos=valor[9]/4;
        for (k=0;k<pos;k++)
        {
                i++;
                if (i%3 == 0 && i != 0 && k != pos-1)
                {
```

```
                    j++;
                    i=-1;
                }
        }

        if (codi_operacio == 35) r[rt]=valor[11]=Mem->Cont[i][j];
        if (codi_operacio == 43) Mem->Cont[i][j]=r[rt];
}
```

L'anterior tros de s'encarrega primer de trobar l'adreça correcta on s'ha d'introduir o extraure el valor de la memòria de dades. Si la posició a accedir no és múltiple de quatre, eixirà un missatge per pantalla i l'accés no s'efectuarà. Mentre que si l'accés es pot efectuar, trobarà la posició [i][j] de la matriu *Cont* i emmagatzemarà o extraurà el valor segons siga el codi d'operació.

4.6. Execució numèrica de la instrucció.

Una vegada explicats tots els procediments per poder visualitzar gràficament el bon funcionament del simulador, es passa ara a explicar l'execució numèrica de la instrucció, és a dir al desenvolupament correcte per part del simulador de la instrucció introduïda.

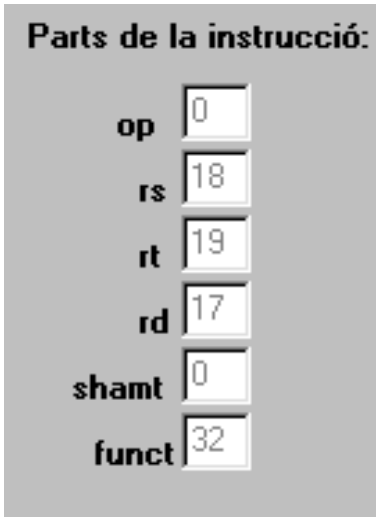
En definitiva es tracta de realitzar l'operació que s'executa a l'ALU. La funció creada ha sigut *Mostrar_Instrucció*, que s'encarrega a més de traure per pantalla els diversos camps dels que consta la instrucció. Podem veure la part corresponent al codi de l'execució de la instrucció *add* a continuació:

```
rs=registre[1];
Ers->Text= AnsiString (rs);
Ers->Visible = 1;
Lrs->Visible = 1;

rt=registre[2];
Ert->Text= AnsiString (rt);
Ert->Visible = 1;
Lrt->Visible = 1;

rd=registre[0];
Erd->Text= AnsiString (rd);
Erd->Visible = 1;
Lrd->Visible = 1;

shamt=0;
Eshamt->Text= AnsiString (shamt);
```

```
Eshamt->Visible = 1;
Lshamt->Visible = 1;

Efunct->Text = AnsiString (funct);
Efunct->Visible = 1;
Lfunct->Visible = 1;

[...]

valor[5]=r[rs]; //Valors per a les linees.
valor[6]=r[rt];
valor[0]=rpc=valor[2];

[...]

if (funct == 32) //add
{
        valor[9]=valor[5]+valor[6];
        strcpy(ALUc,"Suma");
}

[...]

r[rd]=valor[9];
```

El comentat ara es correspon amb la manera de procedir per a les diverses instruccions que poden ser introduïdes al simulador.

Com es veu, primer traurem per pantalla totes les parts de la instrucció, com que la instrucció *add* és de tipus R, el resultat de mostrar les parts de la instrucció es veu a la següent figura:



A més de permetre la visualització de les parts de la instrucció, aquest primer sector de codi, també guarda el nombre de registre que conté cada *parametre* comentat a l'anterior apartat 4.3. del present capítol. Per tant es podrà accedir al contingut del registre que ens fa falta, degut a que aquests *parametres* es corresponen amb les components del vector *r*.

El següent sector de codi guarda a les components del vector *valor* els valors que ha de contindre per visualitzar després amb l'activació de les línies comentada a l'apartat 4.4. Com és de suposar aquest vector *valor* es correspon també amb el vector *Contingut,* (que ens serveix per escriure el text a la pantalla) de la funció *Activar*, de tal manera que com ja hem vist, al activar una línia si aquesta conté text, aquest valor és mostrat per pantalla.

A continuació ja podem executar la instrucció, ja que sabem els continguts dels registres. Al mateix temps s'indica per pantalla que l'ALU va a realitzar l'operació de suma.

Per a finalitzar, es guarda al registre de destí *rd*, el valor de d'aquesta operació.

Cal mencionar que aquesta funció es realitza al prémer per primera vegada el botó de *Següent*, per tant l'execució de la instrucció (numèricament parlant), es realitza abans de realitzar el primer pas de les línies, (es recorda que era la funció Pas1), per tant no és del tot correcte. El que s'ha fet per a evitar que l'usuari introduisca nous valors a la finestra dels registres, degut a que l'operació seria incorrecta, (els valors introduïts de nou no els tindria en compte), ha sigut no poder mostrar, mentre s'executen tots els passos, el contingut tant dels registres com de la memòria.

4.7. Refresc de la pantalla.

Per últim comentarem un problema que sorgia a l'estar executant el simulador en companyia d'altres programes (o finestres) de *Windows*. L'arrel d'aquest problema era que mentre executàvem el simulador no podíem obrir cap altra finestra de *Windows* ni tampoc les pròpies finestres secundàries que teníem al nostre programa, ja que al tornar a l'execució del simulador, les línies activades i el dibuix de l'arquitectura resultaven molt danyats.

Com a solució a aquest problema, es va decidir recórrer a l'event *OnPaint* de la finestra principal. Aquest event s'executa cada vegada que la nostra finestra principal és activa en *Windows*.

Com es veu a continuació el que es realitza a la funció no és més que retrocedir un pas enrere, sempre que l'arquitectura ja haja estat activada, és a dir, sempre que s'haja començat l'execució de la simulació, i després tornar a activar les línies del pas precedent i dibuixar l'arquitectura, i al finalitzar, tornar al pas on ens trobàvem realment abans de la crida a la funció:

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
        ///El rotllo és: si encara no ha estat activada, no dibuixem encara
        ///l'arquitectura.
        if (activat)
        {
                pas--;
                DibuixaArquitectura (Sender);
                if (pas==0) Pas1();
                if (pas>0)
                {
                        Pas1();
                        Activar ();
                }
                pas++;
        }
}
```

Al següent capítol s'executarà una instrucció per poder visualitzar el funcionament del simulador.