Implementación multiciclo del microprocesador R2000.

1. Introducción.
2. Elementos que forman el microprocesador.
3. Funcionamiento y dificultades en la implementación.
4. Posibilidades de aplicación.

1. Introducción.

Nuestro trabajo consiste en la implementación del microprocesador R2000, utilizando programación orientada a objetos con el compilador SDLC++ (Simulador Digital de Lenguaje C++) para Línux.

El uso de la programación orientada a objetos, posibilita que se pueda realizar la simulación, al tratarse cada componente como una variable objeto, que a efectos de memoria es global, y conserva su valor en todo momento.

Si observamos la estructura de un computador, podemos observar que se sigue una jerarquía de componentes, desde unidades simples como pueden las puertas lógicas básicas (and, or, xor ...) a otras complejas, compuestas de las anteriores (sram, alu, fichero de registros ...), pasando por niveles intermedios de complejidad (flip-flops, multiplexores, codificadores ...).

En todo esto se basa nuestro trabajo, en ir simulando circuitos, e ir interconectándolos para lograr lo que llega a ser la unidad u objeto más complejo de todos, que es el microprocesador.

Colorario: INTRODUCCIÓN A LA POO (Programación orientada a objetos).

Un objeto es una encapsulación de un conjunto de datos y de los métodos para manipularlos. Los objetos pueden establecer una comunicación entre ellos. La POO (Programación orientada a objetos) es una forma de programación que utiliza objetos que responden a sucesos. Es un enfoque sumamente natural, puesto que la estructura de los programas refleja directamente la estructura del problema. Además puede considerarse como una extensión natural de la programación estructurada en un intento de potenciar los conceptos de modularidad y reutilización del código.

Los mecanismos básicos de la programación orientada a objetos son: *objetos, mensajes, métodos, clases y subclases.*

MECANISMOS BASICOS PARA LA POO.

OBJETOS

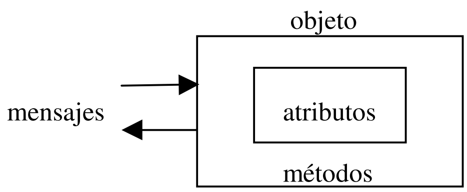
Un programa tradicional se compone de procedimientos y de datos. Un programa orientado a objetos se compone solamente de objetos. Un objeto es una entidad que tiene unos atributos particulares, los datos, y unas formas de operar sobre ellos, los métodos.

MENSAJES

En un programa orientado a objetos, los objetos están recibiendo, interpretando y respondiendo a mensajes de otros objetos. En la POO el flujo de control funcional es reemplazado por el paso de mensajes entre objetos, lo que origina cambio en el estado del objeto. El conjunto de mensajes a los que un objeto puede responder se denomina *protocolo.*
En C++, un mensaje está asociado con el prototipo de una función miembro de tal forma que cuando se produce ese mensaje se ejecuta la correspondiente función miembro de la clase a la que pertenece el objeto. Esto es, el envio de un mensaje equivale en C++ a llamar a una función miembro.

METODOS

Un método (que en C++ se denomina *función miembro*) se implementa en una clase, y determina como tiene que actuar el objeto cuando recibe el mensaje. En adición, las variables asociadas o atributos permitirán almacenar información para dicho objeto. Un método puede también enviar mensajes a otros objetos solicitando una acción o información. La descripción o signatura de un método se denomina operación.
En C++, un método se corresponde con la definición de la función miembro de una clase.



La estructura más interna de un objeto está oculta para otros usuarios y la única conexión que tiene con el exterior son los mensajes. Los atributos (datos) que oculta un objeto, solamente pueden ser manipulados por los métodos asociados al propio objeto.

Según lo expuesto, la ejecución de un programa orientado a objetos realizará fundamentalmente tres cosas:

1. Creará los objetos necesarios.

2. Los mensajes enviados a unos y otros objetos darán lugar a que se procese internamente la información.

3. Finalmente, cuando los objetos no son necesarios, serán borrados, liberándose la memoria ocupada por los mismos.

CLASES

Una clase se puede considerar como una plantilla para crear objetos de esa clase. Una clase es un tipo de objeto definido por el usuario.

CARACTERISTICAS DE LA POO

ABSTRACCION

Abstracción significa " acción de separar mentalmente" o "representación de las características esenciales de algo sin incluir antecedentes o detalles irrelevantes".

ENCAPSULAMIENTO

El encapsulamiento u ocultamiento de información nos permitirá manipular los objetos como unidades básicas, permaneciendo oculta su estructura interna.

VENTAJAS E INCONVENIENTES DE LOS LENGUJES ORIENTADOS A OBJETOS

Un lenguaje de programación que soporte programación orientada a objetos beneficia el desarrollo de software. Esto es así, porque:

- Los objetos bien diseñados son la base para sistemas que se construyen a partir de módulos reutilizables, dando lugar a una mayor productividad.

- La reutilización de las clases que han sido probadas en circunstancias reales, en proyectos anteriores, dan lugar a sistemas de mayor calidad y con menos errores.

- Un diseño orientado a objetos es una herramienta clave para enfrentarse a problemas complejos porque favorece el paso de pequeños a grandes sistemas.

- Las aplicaciones requieren escribir menos lineas de código, menos sentencias de bifurcación y por lo tanto, los módulos son más comprensibles.

- La encapsulación ayuda a construir sistemas más seguros.

- En contraste con la programación tradicional, donde siempre existe la posibilidad de que una modificación de un procedimiento o de una estructura de datos afecte a otros procedimientos, la programación orientada a objetos no necesita examinar todo el código fuente para ver si un cambio local afecta a otras partes.

Según lo expuesto, las ventajas de la programación orientada a objetos son sustanciales. Sin embargo, también presenta algunas desventajas que citamos a continuación:

- La ejecución de una aplicación orientada a objetos resulta algo más lenta.

- Es preciso desarrollar bibliotecas de clases, lo que obliga al usuario a tomar conocimiento de las mismas.

- Hay costes inevitables, asociados al entrenamiento y aprendizaje.

- La administración de bibliotecas de componentes es difícil y costosa.

- Es necesario un cambio en la forma de pensar, y a todo el mundo le sienta mal.

2. Elementos que forman el microprocesador.

Los elementos se han ido implementando siguiendo un orden de menor a mayor complejidad, de forma que los últimos engloban a los primeros, por decirlo de algún modo.

Para cada elemento se ha declarado un objeto o clase, el cual consta de una declaración y de un método.
A continuación describimos los objetos utilizados en orden de complejidad:

- **Para realizar la simulación necesitamos 3 tipos de datos; VCC(tensión), GND(tierra), WIRE(cable), las cuales representamos del siguiente modo:**

```
#define WIRE unsigned char
#define VCC  255
#define GND  0
```

- **También será necesario el uso de un reloj del sistema clock, representado del siguiente modo:**

  Clase:

```
// Generador de onda cuadrada
class CLOCK {
  int duracion_del_semiperiodo;
 public:
  CLOCK(int duracion_del_semiperiodo);
  void run(int iteracion_de_simulacion,
         WIRE &ck);
};
```

```
void Clock(int iteracion_de_simulacion,
           int duracion_del_semiperiodo,
           WIRE &senhal_de_reloj);
```

Método:

```
#include "defs.h"
#include "clock.h"

CLOCK::CLOCK(int semiperiodo) {
  duracion_del_semiperiodo = semiperiodo;
}

void CLOCK::run(int iteracion, WIRE &ck) {
  if(ck<127) ck=GND; else ck=VCC;
  if(!(iteracion%duracion_del_semiperiodo)) ck = ~ck;
}

void Clock(int iter, int duracion, WIRE &io) {
  if(io<127) io=0; else io=255;
  if(!(iter%duracion)) io = ~io;
}
```

- ## Las puertas lógicas:

Clase:

```
class NOT {
  WIRE delay;
public:
  void run(WIRE &i, WIRE &o);
};

class AND {
  WIRE delay;
public:
  void run(WIRE &i0, WIRE &i1, WIRE &o);
  //void run(WIRE &i0, WIRE &i1, WIRE &i2, WIRE &o);
  void run(WIRE &out, int inputs, WIRE &in, ...);
  void run(WIRE &out, int inputs, WIRE in[]);
};

class NAND {
  WIRE delay;
public:
  void run(WIRE &i0, WIRE &i1, WIRE &o);
  //void run(WIRE &i0, WIRE &i1, WIRE &i2, WIRE &o);
  void run(WIRE &out, int inputs, WIRE &in, ...);
  void run(WIRE &out, int inputs, WIRE in[]);
};

class OR {
  WIRE delay;
public:
  void run(WIRE &i0, WIRE &i1, WIRE &o);
  //void run(WIRE &i0, WIRE &i1, WIRE &i2, WIRE &o);
  void run(WIRE &out, int inputs, WIRE &in, ...);
  void run(WIRE &out, int inputs, WIRE in[]);
};

class NOR {
```

```
  WIRE delay;
public:
  void run(WIRE &i0, WIRE &i1, WIRE &o);
  //void run(WIRE &i0, WIRE &i1, WIRE &i2, WIRE &o);
  void run(WIRE &out, int inputs, WIRE &in, ...);
  void run(WIRE &out, int inputs, WIRE in[]);
};

class XOR {
  WIRE delay;
public:
  void run(WIRE &i0, WIRE &i1, WIRE &o);
  //void run(WIRE &i0, WIRE &i1, WIRE &i2, WIRE &o);
  void run(WIRE &out, int inputs, WIRE &in, ...);
  void run(WIRE &out, int inputs, WIRE in[]);
};

class NXOR {
  WIRE delay;
 public:
  void run(WIRE &i0, WIRE &i1, WIRE &o);
  //void run(WIRE &i0, WIRE &i1, WIRE &i2, WIRE &o);
  void run(WIRE &out, int inputs, WIRE &in, ...);
  void run(WIRE &out, int inputs, WIRE in[]);
};

class BUFFER_Z {
  WIRE delay[2];
  int sample;
public:
  BUFFER_Z();
  void run(WIRE &i, WIRE &e, WIRE &o);
};
```

Método:

```
#include <stdarg.h>
#include "defs.h"
#include "gates.h"

void NOT::run (WIRE &i, WIRE &o) {
  WIRE tmp = ~i;
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void AND::run (WIRE &i0, WIRE &i1, WIRE &o) {
  WIRE tmp = i0 & i1;
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void AND::run(WIRE &out, int inputs, WIRE &in, ...) {
  va_list args;                 // Puntero a la lista de argumentos
  va_start(args,in);            // que ahora apunta al primero deellos
  WIRE tmp = in;                // Primera entrada
  while(--inputs) {             // Para el resto de entradas
    tmp &= va_arg(args,WIRE);   // AND con la siguiente entrada
  }
  va_end(args);                 // Finalizaci'on de manipulaci'on
```

```
  if(delay==tmp) {               // Miramos si la salida es estable
    if(tmp<127) out=GND;         // y binarizamos
    else out=VCC;
  }
  delay=tmp;                     // Recordamos la "posible salida"
}

void AND::run(WIRE &o, int inputs, WIRE i[]) {
  WIRE tmp = i[0];
  for(int j=1;j<inputs;j++) {
    tmp &= i[j];
  }
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void NAND::run (WIRE &i0, WIRE &i1, WIRE &o) {
  WIRE tmp = ~(i0 & i1);
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void NAND::run(WIRE &out, int inputs, WIRE &in, ...) {
  va_list args;                  // Puntero a la lista de argumentos
  va_start(args,in);             // que ahora apunta al primero de
ellos
  WIRE tmp = in;                 // Primera entrada
  while(--inputs) {              // Para el resto de entradas
    tmp &= va_arg(args,WIRE);    // AND con la siguiente entrada
  }
  va_end(args);                  // Finalizaci'on de manipulaci'on
  tmp = ~tmp;                    // Negamos la AND
  if(delay==tmp) {               // Miramos si la salida es estable
    if(tmp<127) out=GND;         // y binarizamos
    else out=VCC;
  }
  delay=tmp;                     // Recordamos la "posible salida"
}

void NAND::run(WIRE &o, int inputs, WIRE i[]) {
  WIRE tmp = i[0];
  for(int j=1;j<inputs;j++) {
    tmp &= i[j];
  }
  tmp = ~tmp;
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void OR::run(WIRE &i0, WIRE &i1, WIRE &o) {
  WIRE tmp = i0 | i1;
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void OR::run(WIRE &out, int inputs, WIRE &in, ...) {
  va_list args;                  // Puntero a la lista de argumentos
```

```
  va_start(args,in);             // que ahora apunta al primero de
ellos
  WIRE tmp = in;                 // Primera entrada
  while(--inputs) {              // Para el resto de entradas
    tmp |= va_arg(args,WIRE);    // OR con la siguiente entrada
  }
  va_end(args);                  // Finalizaci'on de manipulaci'on
  if(delay==tmp) {               // Miramos si la salida es estable
    if(tmp<127) out=GND;         // y binarizamos
    else out=VCC;
  }
  delay=tmp;                     // Recordamos la "posible salida"
}

void OR::run(WIRE &o, int inputs, WIRE i[]) {
  WIRE tmp = i[0];
  for(int j=1;j<inputs;j++) {
    tmp |= i[j];
  }
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void NOR::run (WIRE &i0, WIRE &i1, WIRE &o) {
  WIRE tmp = ~(i0 | i1);
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}


void NOR::run(WIRE &out, int inputs, WIRE &in, ...) {
  va_list args;                  // Puntero a la lista de argumentos
  va_start(args,in);             // que ahora apunta al primero de
ellos
  WIRE tmp = in;                 // Primera entrada
  while(--inputs) {              // Para el resto de entradas
    tmp |= va_arg(args,WIRE);    // OR con la siguiente entrada
  }
  va_end(args);                  // Finalizaci'on de manipulaci'on
  tmp = ~tmp;                    // Negamos la OR
  if(delay==tmp) {               // Miramos si la salida es estable
    if(tmp<127) out=GND;         // y binarizamos
    else out=VCC;
  }
  delay=tmp;                     // Recordamos la "posible salida"
}

void NOR::run(WIRE &o, int inputs, WIRE i[]) {
  WIRE tmp = i[0];
  for(int j=1;j<inputs;j++) {
    tmp |= i[j];
  }
  tmp = ~tmp;
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void XOR::run (WIRE &i0, WIRE &i1, WIRE &o) {
```

```
  WIRE tmp = i0 ^ i1;
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void XOR::run(WIRE &out, int inputs, WIRE &in, ...) {
  va_list args;                 // Puntero a la lista de argumentos
  va_start(args,in);            // que ahora apunta al primero de
ellos
  WIRE tmp = in;                // Primera entrada
  while(--inputs) {             // Para el resto de entradas
    tmp ^= va_arg(args,WIRE);   // XOR con la siguiente entrada
  }
  va_end(args);                 // Finalizaci'on de manipulaci'on
  if(delay==tmp) {              // Miramos si la salida es estable
    if(tmp<127) out=GND;        // y binarizamos
    else out=VCC;
  }
  delay=tmp;                    // Recordamos la "posible salida"
}

void XOR::run(WIRE &o, int inputs, WIRE i[]) {
  WIRE tmp = i[0];
  for(int j=1;j<inputs;j++) {
    tmp ^= i[j];
  }
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void NXOR::run (WIRE &i0, WIRE &i1, WIRE &o) {
  WIRE tmp = ~(i0 ^ i1);
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

void NXOR::run(WIRE &out, int inputs, WIRE &in, ...) {
  va_list args;                 // Puntero a la lista de argumentos
  va_start(args,in);            // que ahora apunta al primero de
ellos
  WIRE tmp = in;                // Primera entrada
  while(--inputs) {             // Para el resto de entradas
    tmp ^= va_arg(args,WIRE);   // XOR con la siguiente entrada
  }
  va_end(args);                 // Finalizaci'on de manipulaci'on
  tmp = ~tmp;                   // Negamos la XOR
  if(delay==tmp) {              // Miramos si la salida es estable
    if(tmp<127) out=GND;        // y binarizamos
    else out=VCC;
  }
  delay=tmp;                    // Recordamos la "posible salida"
}

void NXOR::run(WIRE &o, int inputs, WIRE i[]) {
  WIRE tmp = i[0];
  for(int j=1;j<inputs;j++) {
    tmp ^= i[j];
  }
```

```
  tmp = ~tmp;
  if(delay==tmp) o=tmp;
  delay=tmp;
  if(o<127) o=GND; else o=VCC;
}

BUFFER_Z::BUFFER_Z(){sample=0;}
void BUFFER_Z::run(WIRE &i, WIRE &e, WIRE &o) {
  delay[sample] = i;
  sample = (sample+1) % 2;
  if(delay[0]!=delay[1]) return;
  if(e==VCC) o=delay[0];
}
```

- **Utilizaremos varios tipos de multiplexores:**

  Clase:

```
class MUX_2_1 {
  NOT not;
  BUFFER_Z buf[2];
  WIRE nsel;
 public:
  void run(WIRE in[2], WIRE &sel, WIRE &out);
  void run(WIRE &in0, WIRE &in1, WIRE &sel, WIRE &out);
};

/*
 * Demultiplexor de 1:2
 *
 *     sel
 *      |
 *      +--NOT--A
 *      |       N-- out[0]
 *      |    +--D
 *  in--|----+
 *      |    +--A
 *      |       N-- out[1]
 *      +-------D
 */
class DEMUX_1_2 {
  NOT not;
  AND and[2];
  WIRE nsel;
 public:
  void run(WIRE &in, WIRE &sel, WIRE out[2]);
};

/* Mux 4:1
 *
 *          sel[2]
 *            |
 *        +-------+
 *      / DECOD_2_4 \ dec
 *     +-------------+
 *   d[4]|  |  |  |d[0]
 *       |  |  |  +
 *       |  |  | | \ buf[4]
 *    -----------+  +---+
 *       |  |  | | /    |
 *       |  |  +  +     |
 *       |  | | \       |
 *    --------+  +------+
 *       |  | | /       |
 * in[4] |  +  +        +--- out
 *       | | \          |
 *    -----+  +---------+
 *       | | /          |
```

```
 *      +  +          |
 *     | \            |
 *   --+  +------------+
 *     | /
 *      +
 */

class MUX_4_1 {
  DECOD_2_4 dec;
  BUFFER_Z buf[4];
  WIRE d[4];

public:
  void run(WIRE in[4], WIRE sel[2], WIRE &out);
  void run(WIRE &in0, WIRE &in1, WIRE &in2, WIRE &in3, WIRE sel[2],
WIRE &out);
};


/*
 * Un demultiplexor de 4:1 es un circuito similar en el que existe un
 * decodificador 2:4 y 4 puertas AND colocadas en las 4 salidas.
 * La entrada al demultiplexor entra a cada una de las puertas.
 * Las 4 salidas del demultiplexor son las 4 puertas AND.
 */
class DEMUX_1_4 {
  DECOD_2_4 dec;
  AND and[4];
  WIRE d[4];

 public:
  void run(WIRE &in, WIRE sel[2], WIRE out[4]);
};

/*
 * Multiplexor de tama~no variable.
 */
class MUX{
  DECOD decoder;
  BUFFER_Z *buffers;
  WIRE *outs_decoder;
  int num_ins;         // N'umero de entradas (no hardware)
 public:
  void create(int size);
  void destroy();
  void run(WIRE in[], WIRE sel[], WIRE &out);
};

/*
 * Demultiplexor de tama~no variable.
 */
class DEMUX {
  DECOD decoder;
  AND *and;
  WIRE *out_decoder;
  int num_outs;

 public:
  DEMUX();
  DEMUX(int num_sels);
  void create(int num_sels);
  void destroy();
  void run(WIRE &in, WIRE sel[], WIRE out[]);
```

```
};
```

Método:

```
#include "defs.h"
#include "gates.h"
#include "decods.h"
#include "muxs.h"

void MUX_2_1::run(WIRE in[2], WIRE &sel, WIRE &out) {
  not.run(sel,nsel);
  buf[0].run(in[0],nsel,out);
  buf[1].run(in[1],sel,out);
}
void MUX_2_1::run(WIRE &in0, WIRE &in1, WIRE &sel, WIRE &out) {
  not.run(sel,nsel);
  buf[0].run(in0,nsel,out);
  buf[1].run(in1,sel,out);
}

void DEMUX_1_2::run(WIRE &in, WIRE &sel, WIRE out[2]) {
  not.run(sel,nsel);
  and[0].run(nsel,in,out[0]);
  and[1].run(sel,in,out[1]);
}

void MUX_4_1::run(WIRE in[4], WIRE sel[2], WIRE &out) {
  dec.run(sel,d);
  for(int i=0;i<4;i++)
    buf[i].run(in[i],d[i],out);
}

void MUX_4_1::run(WIRE &in0,WIRE &in1,WIRE &in2,WIRE &in3, WIRE
sel[2], WIRE &out) {
  dec.run(sel,d);
  buf[0].run(in0,d[0],out);
  buf[1].run(in1,d[1],out);
  buf[2].run(in2,d[2],out);
  buf[3].run(in3,d[3],out);
}
void DEMUX_1_4::run(WIRE &in, WIRE sel[2], WIRE out[4]) {
  dec.run(sel,d);
  for(int i=0;i<4;i++)
    and[i].run(d[i],in,out[i]);
}

void MUX::create(int num_sels) {
  decoder.create(num_sels);
  num_ins = 1<<num_sels;
  buffers = new BUFFER_Z[num_ins];
  outs_decoder = new WIRE[num_ins];
}

void MUX::destroy() {
  decoder.destroy();
  delete buffers;
  delete outs_decoder;
}

void MUX::run(WIRE in[], WIRE sel[], WIRE &out) {
  decoder.run(sel,outs_decoder);
```

```
  for(int i=0; i<num_ins; i++) {
    buffers[i].run(in[i],outs_decoder[i],out);
  }
}

DEMUX::DEMUX() {}
void DEMUX::create(int num_sels) {
  decoder.create(num_sels);
  num_outs = 1<< num_sels;
  and = new AND[num_outs];
  out_decoder = new WIRE[num_outs];
}
void DEMUX::destroy() {
  decoder.destroy();
  delete and;
  delete out_decoder;
}
DEMUX::DEMUX(int num_sels) {
  create(num_sels);
}
void DEMUX::run(WIRE &in, WIRE sel[], WIRE out[]) {
  decoder.run(sel,out_decoder);
  for(int i=0;i<num_outs;i++)
    and[i].run(out_decoder[i],in,out[i]);
}
```

- **Flip-flops:**

  Clase:

```
/*
 * Detector de flancos positivos.
 * Genera un pulso positivo de duraci'on igual a tres puertas
 * NOT tras la llegada de un flanco positivo.
 *
 * Este dispositivo es muy usado en la construcci'on de
 * flip-flops a partir de cerrojos.
 *
 *    i   +----+ a0 +----+ a1 +----+ a2 +--
 * ---+---| n0 |----| n1 |----| n2 |----|   \  o
 *    |   +----+    +----+    +----+    | na |---
 *    +---------------------------------|   /
 *                                      +--
 */

class POSITIVE_DETECTOR {
  NOT n0,n1,n2; // Retardo + negaci'on
  AND an;
  WIRE a0,a1,a2;
 public:
  void run(WIRE &i, WIRE &o);
};

/*
 * Eliminador de "gliches" positivos.
 *
 *                             +------+
 * a ---+----------------------|      |
 *      | +-----+   +-----+    | AND  |---- d
 *      +-| NOT |---| NOT |---|      |
 *        +-----+ b +-----+ c +------+
 *
 *          ++     +-----
 * a        ||     |
 *          ||     |
 *   -------++-----+
 *    -------++-----+
```

```
 *          ||     |
 * b        ||     |
 *          ++     +-----
 *           ++     +-----
 * c         ||     |
 *           ||     |
 *     -------++-----+
 *                     +-----
 *                     |
 * d                   |
 *      --------------+
 * La se~nal de salida est'a retarda 6 ciclos respecto de la de
 * entrada, pero no presenta "gliches" positivos.
 */
class POSITIVE_GLICH_SUPRESSOR {
  NOT not[2];
  AND and;
  WIRE b,c;
 public:
  void run(WIRE &a, WIRE &d);
};


/*
 * Eliminador de "gliches" negativos.
 *
 *                             +-----+
 * a ---+---------------------|     |
 *      | +-----+   +-----+   |  OR |---- d
 *      +-| NOT |---| NOT |---|     |
 *        +-----+ b +-----+ c +-----+
 *
 *   -------++-----+
 * a        ||     |
 *          ||     |
 *          ++     +-----
 *           ++     +-----
 *           ||     |
 * b         ||     |
 *    -------++-----+-----
 *     -------++-----+-----
 * c          ||     |
 *            ||     |
 *            ++     +-----
 *      --------------+
 *                     |
 * d                   |
 *                     +-----
 * La se~nal de salida est'a retarda 6 ciclos respecto de la de
 * entrada, pero no presenta "gliches" negativos.
 */
class NEGATIVE_GLICH_SUPRESSOR {
  NOT not[2];
  OR or;
  WIRE b,c;
 public:
  void run(WIRE &a, WIRE &d);
};

/*
 * Se construye negando la entrada a un detector de flancos positivo.
 *
 *      +-----+ ni +----------+
 *  i --| NOT |----| POSITIVE |-- o
 *      +-----+    | DETECTOR |
 *                 +----------+
 */

class NEGATIVE_DETECTOR {
  POSITIVE_DETECTOR pd;
  NOT no;
```

```
  WIRE ni;
 public:
  void run(WIRE &i, WIRE &o);
};

/*
 * Flip-flop SR disparado por flanco negativo.
 * Se trata de un cerrojo SR con entrada de habilitaci'on, que se
 * habilita s'olo durante el tiempo que dura el detector de flancos.
 *                       +-----+
 *      +----------+  -|S     |-- Q
 * ck --| NEGATIVE |---|ENAB |
 *      | DETECTOR |  -|R     |O- Qn
 *      +----------+   +-----+
 */

class NEG_SR_FF {
  NEGATIVE_DETECTOR detec;
  SR_LATCH_ENABLE latch;
  WIRE pulse;
public:
  void run(WIRE &S, WIRE &R, WIRE &ck, WIRE &Q, WIRE &Qn);
};

/*
 * Flip-flop D disparado por flanco negativo.
 *
 *          +-----+
 *  D --+---|S    |-- Q
 * ck --|--O|>    |
 *      +-O-|R    |O- Qn
 *          +-----+
 * (el circulo en medio de la l'inea representa una puerta NOT).
 */

class NEG_D_FF {
  NEG_SR_FF ff;
  NOT n;
  WIRE Dn;
 public:
  void run(WIRE &D, WIRE &ck, WIRE &Q, WIRE &Qn);
};

/*
 * Flip-flop JK disparado por flanco negativo.
 *
 *                       +-----+
 *      +----------+  -|J     |-- Q
 * ck --| NEGATIVE |---|ENAB |
 *      | DETECTOR |  -|K     |O- Qn
 *      +----------+   +-----+
 */

class NEG_JK_FF {
  NEGATIVE_DETECTOR dec;
  WIRE pulse;
  JK_LATCH_ENABLE latch_JK;
public:
  void run(WIRE &J, WIRE &K, WIRE &ck, WIRE &Q, WIRE &Qn);
};

/*
 * Flip-flop JK disparado por flanco negativo, con entradas de PReseT y
 * CLeaR activas en bajo.
 *
 *                          +-----------------------------------------+
 *                          |                 PRT                     |
 *                          |+------------+    |                      |
 *                          ||  +------+  |    |    +------+          |
```

```
 *                         |+--|      |  | a +---|      |            |
 *                         +---| NAND |O-|---|---| NAND |O--+-+- Q   |
 *  J -------------------------| na0  |  |   / +-| na2  |  /  |      |
 *        +----------+      +--|      |  \  /  | +-------+ /   |      |
 * ck ____| NEGATIVE |______|  +-------+   \/  +|----------    |      |
 *        | DETECTOR | pulse|  +-------+   /\  |+----------    |      |
 *        +----------+      +--|      |  /  \ |  +-------+ \   |      |
 *  K -------------------------| NAND |  | b \+--|      |  \  |      |
 *                         +---| na1  |O-|---|---| NAND |O--+-|- Qn  |
 *                         |+--|      |  |   +---| na3  |   |  |      |
 *                         ||  +-------+  |   |   +-------+   +-|------+
 *                         |+-------------+   |                |
 *                         |                CLR               |
 *                         +----------------------------------+
 *
 * fig 8-26. Fundamentos de Sistemas Digitales.
 * T. L. Floyd.
 */

class NEG_JK_FF_PC {
  NEGATIVE_DETECTOR dec;
  NAND na0,na1,na2,na3;
  WIRE pulse,a,b;
public:
  void run(WIRE &J, WIRE &K, WIRE &ck, WIRE &PRT,
           WIRE &CLR, WIRE &Q, WIRE &Qn);
};

/*
 * Flip-flip SR maestro-esclavo, disparado por flanco negativo.
 *
 *           master     slave
 *          +-----+    +-----+
 *    S-----|S     |----|S     |-- Q
 *        +-|ENAB  |  +-|ENAB  |
 *    R---|-|R     |O-|-|R     |O- Qn
 *        | +-----+  | +-----+
 *        | +-----+  |
 *   ck---+-| NOT |--+
 *          +-----+
 */

class SR_FF_MS {
  SR_LATCH_ENABLE master,slave;
  NOT n;
  WIRE nck,q,qn;
public:
  void run(WIRE &S, WIRE &R, WIRE &ck, WIRE &Q, WIRE &Qn);
};

/*
 * Flip-flop JK maestro-esclavo, disparado por flanco negativo.
 *
 *   +-----------------------+
 *   | +----+                |
 *   +-| AND| a +-------+    |
 *  J--|  a0|---|S    -+|--+-|- Q
 *     +----+   |      ||  | |
 *  ck----------|       |  | |
 *     +----+ b |     -+|  | |
 *  K--| AND|---|R     ||O-|-+- Qn
 *   +-|  A1|   +-------+  |
 *   | +----+              |
 *   +---------------------+
 */

class JK_FF_MS {
  SR_FF_MS ff;
  AND a0,a1;
  WIRE a,b;
public:
```

```
  void run (WIRE &J, WIRE &K, WIRE &ck, WIRE &Q, WIRE &Qn);
};
```

Método:

```
#include "defs.h"
#include "gates.h"
#include "latches.h"
#include "flip-flops.h"

void POSITIVE_DETECTOR::run(WIRE &i, WIRE &o) {
  n0.run(i,a0);
  n1.run(a0,a1);
  n2.run(a1,a2);
  an.run(i,a2,o);
}

void NEGATIVE_DETECTOR::run(WIRE &i, WIRE &o) {
  no.run(i,ni);
  pd.run(ni,o);
}

void POSITIVE_GLICH_SUPRESSOR::run(WIRE &a, WIRE &d) {
  not[0].run(a,b);
  not[1].run(b,c);
  and.run(a,c,d);
}

void NEGATIVE_GLICH_SUPRESSOR::run(WIRE &a, WIRE &d) {
  not[0].run(a,b);
  not[1].run(b,c);
  or.run(a,c,d);
}

void NEG_SR_FF::run(WIRE &S, WIRE &R, WIRE &ck, WIRE &Q, WIRE &Qn) {
  detec.run(ck,pulse);
  latch.run(S,R,pulse,Q,Qn);
}

void NEG_D_FF::run(WIRE &D, WIRE &ck, WIRE &Q, WIRE &Qn) {
  n.run(D,Dn);
  ff.run(D,Dn,ck,Q,Qn);
}


void NEG_JK_FF::run(WIRE &J, WIRE &K, WIRE &c, WIRE &Q, WIRE &Qn) {
  dec.run(c,pulse);
  latch_JK.run(J,K,pulse,Q,Qn);
}

void NEG_JK_FF_PC::run(WIRE &J, WIRE &K, WIRE &c, WIRE &PRT,
                       WIRE &CLR, WIRE &Q, WIRE &Qn) {
  dec.run(c,pulse);
  na0.run(a,4,CLR,Qn,J,pulse);
  na1.run(b,4,PRT, Q,K,pulse);
  na2.run( Q,3,Qn,PRT,a);
  na3.run(Qn,3, Q,CLR,b);
}

void SR_FF_MS::run(WIRE &S, WIRE &R, WIRE &ck, WIRE &Q, WIRE &Qn) {
```

```
  n.run(ck,nck);
  master.run(S,R,ck,q,qn);
  slave.run(q,qn,nck,Q,Qn);
}

void JK_FF_MS::run (WIRE &J, WIRE &K, WIRE &ck, WIRE &Q, WIRE &Qn) {
  a0.run(J,Qn,a);
  a1.run(K,Q,b);
  ff.run(a,b,ck,Q,Qn);
}
```

- ## Decodificadores:

Clase:

```
/* Decodificador 2:4
 *
 * in[1] in[0]   nin[1] nin[0]
 *   |     |        |      |
 *   +-----|-not[1]-+      |
 *   |     +-not[0]-|------+
 *   |     |        |      | +---+
 *   |     |        +------|-|and|___ out[0] = nin[1] & nin[0]
 *   |     |        |      +-|[0]|
 *   |     |        |      | +---+
 *   |     |        |      | +---+
 *   |     +--------|------|-|and|___ out[1] = nin[1] & in[0]
 *   |     |        +------|-|[1]|
 *   |     |        |      | +---+
 *   |     |        |      | +---+
 *   +-----|--------|------|-|and|___ out[2] = in[1] & nin[0]
 *   |     |        |      +-|[2]|
 *   |     |        |      | +---+
 *   |     |        |      | +---+
 *   +-----|--------|------|-|and|___ out[3] = in[1] & in[0]
 *   |     +--------|------|-|[3]|
 *   |     |        |      | +---+
 */

class DECOD_2_4 {
  WIRE nin[2];
  NOT not[2];
  AND and[4];
public:
 void run(WIRE in[2], WIRE out[4]);
};

/*   in[4-0]       nin[4-0]
 *  | | | | |     | | | | |
 *  | | | | +-NOT-|-|-|-|-+
 *  | | | +-|-NOT-|-|-|-+ |
 *  | | +-|-|-NOT-|-|-+ | |
 *  | +-|-|-|-NOT-|-+ | | |
 *  +-|-|-|-|-NOT-+ | | | |
 *  | | | | |     +-|-|-|-|-A
 *  | | | | |     | +-|-|-|-N
 *  | | | | |     | | +-|-|-D--- out[0]
 *  | | | | |     | | | +-|-[0
 *  | | | | |     | | | | +-]
 *  : : : : :     : : : : :
 *  | | | | |
 *  +-|-|-|-|---------------A
 *    +-|-|-|---------------N
 *      +-|-|---------------D--- out[31]
 *        +-|---------------[31
 *          +---------------]
 */

class DECOD_5_32 {
  WIRE nin[5];
```

```
  NOT not[5];
  AND and[32];
 public:
  void run(WIRE in[5], WIRE out[32]);
};

/*
 * Decodificador 2:4 con entrada de selecci'on.
 * Se trata de un decodificador normal que en cada salida tiene una AND
 * cuyas entradas son una de las salidas del decodificador normal y la
 * entrada de selecci'on. Este tipo de decodificador es muy 'util en la
 * construcci'on de memorias.
 */

class DECOD_2_4_SEL {
  DECOD_2_4 dec;
  AND and[4];
  WIRE o[4];
 public:
  void run(WIRE in[2], WIRE &select, WIRE out[4]);
};

class DECOD_5_32_SEL {
  DECOD_5_32 dec;
  AND and[32];
  WIRE o[32];
 public:
  void run(WIRE in[5], WIRE &select, WIRE out[32]);
};

/*
 * Decodificador de tama~no variable.
 */

class DECOD {
  WIRE *nin;
  NOT *not;
  AND *and;
  int num_sels; // N'umero de entradas de selecci'on (no hardware)
  WIRE *aux;    // Array auxiliar que no forma parte de la
descripci'on
                // real del decodificador (no hardware)
 public:
  DECOD();
  DECOD(int num_sels);
  ~DECOD();
  void create(int num_sels); // Asigna memoria a los componentes
internos
  void destroy();            // Libera la memoria
  void run(WIRE *in, WIRE *out); // Simula el decodificador
};
```

Método:

```
#include "defs.h"
#include "gates.h"
#include "decods.h"

void DECOD_2_4::run(WIRE in[2], WIRE out[4]) {
  not[0].run(in[0],nin[0]);
  not[1].run(in[1],nin[1]);
  and[0].run(nin[1],nin[0],out[0]);
  and[1].run(nin[1], in[0],out[1]);
```

```
  and[2].run( in[1],nin[0],out[2]);
  and[3].run( in[1], in[0],out[3]);
}

void DECOD_5_32::run(WIRE in[5], WIRE out[32]) {
  int i;
  for(i=0;i<5;i++) {
    not[i].run(in[i],nin[i]);
  }
  WIRE aux[5];              // Este bus temporal se usa para elegir
  for(i=0;i<32;i++) {       // las entradas correspondientes a cada
    int j;                  // puerta AND.
    for(j=0;j<5;j++) {      // Este bucle lo que hace b'asicamente es
      if((i>>j)%2) {        // generar un n'umero binario de 5 bits.
        aux[j]=in[j];       // Y en funci'on de que el bit sea 0 o 1,
      } else {              // seleccionamos nin[j] o in[j]
        aux[j]=nin[j];      // respectivamente.
      }
    }
    and[i].run(out[i],5,aux);
  }
}

void DECOD_2_4_SEL::run(WIRE in[2], WIRE &select, WIRE out[4]) {
  dec.run(in,o);                      // Simulamos el decodificador 2:4
  for(int i=0;i<4;i++)
    and[i].run(o[i],select,out[i]);  // Simulamos las 4 AND
}

void DECOD_5_32_SEL::run(WIRE in[5], WIRE &select, WIRE out[32]) {
  dec.run(in,o);                      // Simulamos el decodificador
5:32
  for(int i=0;i<32;i++) {
    and[i].run(o[i],select,out[i]);  // Simulamos las 32 AND
  }
}

void DECOD::create(int num_sels) {
  this->num_sels = num_sels;
  int num_outs = 1<<num_sels;
  nin = new WIRE[num_sels];
  not = new NOT[num_sels];
  and = new AND[num_outs];
  aux = new WIRE[num_sels]; // Variable auxiliar
}
void DECOD::destroy() {
  delete nin;
  delete not;
  delete and;
  delete aux;
}
DECOD::DECOD(){}
DECOD::DECOD(int num_sels) {
  create(num_sels);
}
DECOD::~DECOD() {
  destroy();
}
void DECOD::run(WIRE *in, WIRE *out) {
  int i;
  int num_outs = 1<<num_sels;
```

```
  for(i=0;i<num_sels;i++)
    not[i].run(in[i],nin[i]);
  //WIRE *aux = new WIRE[num_sels];
  //WIRE aux[num_sels];
  for(i=0;i<num_outs;i++) {
    for(int j=0;j<num_sels;j++) {
      if((i>>j)&0x01) {
        aux[j]=in[j];
      } else {
        aux[j]=nin[j];
      }
    }
    and[i].run(out[i],num_sels,aux);
  }
}
```

- **Cerrojos:**

  Clase:

```
/*
 * Dos puertas NOR realimentadas de la siguiente forma,
 *  implementan un cerrojo SR:
 *
 *            +------+
 *      R ----+      |
 *            | NOR  +--+-- Q
 *          +-+  n0  |  |         S R | Q Qn
 *          | +------+ /         -----+-----
 *           \       /            0 0 | Q Qn (estado anterior)
 *             \   /              0 1 | 0 1  (reset)
 *               X                1 0 | 1 0  (set)
 *             /   \              1 1 | 0 0  (estado indeseable)
 *           /       \
 *          | +------+ \
 *          +-+      |  |
 *            | NOR  +--+-- Qn
 *      S ----+  n1  |
 *            +------+
 */
class SR_LATCH_NOR {
  NOR n0,n1;
public:
  void run(WIRE &S, WIRE &R, WIRE &Q, WIRE &Qn);
};

/*
 *            +------+
 *      R ----+      |
 *            | NAND +--+-- Q
 *          +-+  n0  |  |         S R | Q Qn
 *          | +------+ /         -----+-----
 *           \       /            0 0 | 1 1  (estado indeseable)
 *             \   /              0 1 | 0 1  (reset)
 *               X                1 0 | 1 0  (set)
 *             /   \              1 1 | Q Qn (estado anterior)
 *           /       \
 *          | +------+ \
 *          +-+      |  |
 *            | NAND +--+-- Qn
 *      S ----+  n1  |
 *            +------+
 */
class SR_LATCH_NAND {
  NAND n0,n1;
public:
  void run(WIRE &S, WIRE &R, WIRE &Q, WIRE &Qn);
};
```

```
/*
 * Se implementa mediante un cerrojo SR-NOR haciendo S=D y R=~D.
 * Este circuito se utiliza muchas veces, adem'as de como elemento de memoria, como
 * circuito de retardo, puesto que Q=D un cierto tiempo despu'es.
 */
class D_LATCH {
  NOT n;
  SR_LATCH_NOR l;
  WIRE Dn;
public:
  void run(WIRE &D, WIRE &Q, WIRE &Qn);
};

/* Se implementa con 4 puertas NAND colocadas de la siguiente forma:
 *
 *                a
 * J ---- NAND ---- NAND -+-- Q   J K | Q Qn
 *        | n0        \   |       ----+-----
 *        +------------\ /        0 0 | Q Qn
 *                      X         0 1 | 0 1
 *        +------------/ \        1 0 | 1 0
 *        | n1        /   |       1 1 | 1 1
 * K ---- NAND ---- NAND -+-- Qn
 *                b
 */

class JK_LATCH {
  NAND na0,na1,na2,na3;
  WIRE a,b;
public:
  void run(WIRE &J, WIRE &K, WIRE &Q, WIRE &Qn);
};

/*
 * Cerrojo JK con entrada de habilitaci'on.
 * Id'entico al anterior salvo porque las puertas de entrada para J y K
 * poseen una tercera entrada a la que entra la se~nal de habilitaci'on E.
 */

class JK_LATCH_ENABLE {
  NAND na0,na1;
  NAND na2,na3;
  WIRE a,b;
public:
  void run(WIRE &J, WIRE &K, WIRE &E, WIRE &Q, WIRE &Qn);
};

/* Cerrojo SR con entrada de habilitaci'on.
 *
 *              a +------+
 *  S -- NAND ---|R      |---- Q
 *       |        |      |
 *  E ---+        |      |
 *       |      b |      |
 *  R -- NAND ---|S      |O--- Q'
 *                +------+
 */

class SR_LATCH_ENABLE {
  SR_LATCH_NAND latch;
  NAND na0,na1;
  WIRE a,b;
public:
  void run(WIRE &S, WIRE &R, WIRE &E, WIRE &Q, WIRE &Qn);
};
```

Metodo:

```
#include "defs.h"
#include "gates.h"
#include "latches.h"

void SR_LATCH_NOR::run(WIRE &S, WIRE &R, WIRE &Q, WIRE &Qn) {
  n0.run(R,Qn,Q);
  n1.run(S,Q,Qn);
}

void SR_LATCH_NAND::run(WIRE &S, WIRE &R, WIRE &Q, WIRE &Qn) {
  n0.run(R,Qn,Q);
  n1.run(S,Q,Qn);
}

void D_LATCH::run(WIRE &D, WIRE &Q, WIRE &Qn) {
  n.run(D,Dn);
  l.run(D,Dn,Q,Qn);
}

void JK_LATCH::run(WIRE &J, WIRE &K, WIRE &Q, WIRE &Qn) {
  na0.run(J,Qn,a);
  na1.run(K,Q,b);
  na2.run(a,Qn,Q);
  na3.run(b,Q,Qn);
}

void JK_LATCH_ENABLE::run(WIRE &J, WIRE &K, WIRE &E, WIRE &Q, WIRE
&Qn) {
  na0.run(a,3,J,E,Qn);
  na1.run(b,3,K,E,Q);
  na2.run(a,Qn,Q);
  na3.run(b,Q,Qn);
}

void SR_LATCH_ENABLE::run(WIRE &S, WIRE &R, WIRE &E, WIRE &Q, WIRE
&Qn) {
  na0.run(S,E,a);
  na1.run(R,E,b);
  latch.run(b,a,Q,Qn);
}
```

- ## Registros:

  Clase:

```
/*
 * Un registro de 4 bits construido con 4 flip-flops D.
 * La l'inea ck provoca que el registro se carge con el contenido
 * presentado a trav'es del bus de entrada i[], cuando se produce
 * un flanco negativo.
 *
 *  i[0]        i[1]        i[2]        i[3]
 *  |           |           |           |
 *  |  +-----+  |  +-----+  |  +-----+  |  +-----+
 *  +--|D    |O +--|D    |O +--|D    |O +--|D    |O
 * ck-O|>    |----O|>    |----O|>    |----O|>    |
 *     |     |--+  |     |--+  |     |--+  |     |--+
 *     +-----+  |  +-----+  |  +-----+  |  +-----+  |
 *              |           |           |           |
 *              o[0]        o[1]        o[2]        o[3]
 */
class REG_4 {
  NEG_D_FF/*_INPUT_D_DELAYED*/ ff[4];
```

```
  WIRE Qn[4];
 public:
  void run(WIRE &ck, WIRE i[4], WIRE o[4]);
};

/*
 * Un registro de 8 bits construido con 2 registros de 4 bits.
 *
 *      i[0..3]      i[4..7]
 *         |            |
 *     +-------+    +-------+
 *     | REG_4 |    | REG_4 |
 *  ck-O|       |-ck-O|       |
 *     | reg[0]|    | reg[1]|
 *     +-------+    +-------+
 *         |            |
 *      o[0..3]      o[4..7]
 *
 */

class REG_8 {
  REG_4 reg[2];
 public:
  void run(WIRE &ck, WIRE i[8], WIRE o[8]);
};

/*
 * Un registro de 16 bits construido con 2 registros de 8 bits.
 */

class REG_16 {
  REG_8 reg[2];
 public:
  void run(WIRE &ck, WIRE i[16], WIRE o[16]);
};

/*
 * Un registro de 32 bits construido con 2 registros de 16 bits.
 */

class REG_32 {
  REG_16 reg[2];
 public:
  void run(WIRE &ck, WIRE i[32], WIRE o[32]);
};
```

Método:

```
#include "defs.h"
#include "gates.h"
#include "latches.h"
#include "flip-flops.h"
#include "regs.h"

void REG_4::run(WIRE &ck, WIRE i[4], WIRE o[4]) {
  ff[0].run(i[0],ck,o[0],Qn[0]);
  ff[1].run(i[1],ck,o[1],Qn[1]);
  ff[2].run(i[2],ck,o[2],Qn[2]);
  ff[3].run(i[3],ck,o[3],Qn[3]);
}

void REG_8::run(WIRE &ck, WIRE i[8], WIRE o[8]) {
  reg[0].run(ck,i,o);
  reg[1].run(ck,i+4,o+4);
```

```
}

void REG_16::run(WIRE &ck, WIRE i[16], WIRE o[16]) {
  reg[0].run(ck,i,o);
  reg[1].run(ck,i+8,o+8);
}

void REG_32::run(WIRE &ck, WIRE i[32], WIRE o[32]) {
  reg[0].run(ck,i,o);
  reg[1].run(ck,i+16,o+16);
}
```

- **Una alu de 32 bits:**

  Clase:

```
/*
 * Sumador de 1 bit.
 *
 *             carry_in
 *                |                              carry_in a b |sum carry_out
 *                |                              -------------+-------------
 *                |  +---+                              0 0 0 |  0         0
 *    a ----+-----|--|AND|____t0                        0 0 1 |  1         0
 *          |     +--|a0 |    |                         0 1 0 |  1         0
 *          |     |  +---+    |  +---+                  0 1 1 |  0         1
 *          |     |  +---+ t1 +--|OR |                  1 0 0 |  1         0
 *    b ----|--+--|--|AND|__-----| o |---+              1 0 1 |  0         1
 *          |  |  +--|a1 |    +--|   |   |              1 1 0 |  0         1
 *          |  |  |  +---+    |  +---+   |              1 1 1 |  1         1
 *          |  |  |  +---+    |          |
 *          |  +--|--|AND|____|          |
 *          +--|--|--|a2 |    t2         |
 *          |  |  |  +---+               |
 *          |  |  |  +---+               |
 *          |  +--|--|XOR|____           |
 *          +-----|--|x0 | t3 | +---+    |
 *                |  +---+    +-|XOR|____|____ sum
 *                +-------------|x1 |    |
 *                              +---+    |
 *                                   carry_out
 */

class ADDER_BIT {

  WIRE t[4];
  AND a0,a1,a2;
  XOR x0,x1;
  OR o;

 public:
  void run(WIRE &a, WIRE &b,      // Entradas de datos
           WIRE &carry_in,        // Acarreo de entrada
           WIRE &sum,             // Resultado de la suma
           WIRE &carry_out);      // Acarreo de salida
};

/*
 * ALU de 1 bit.
 *
 *      binvert                carry_in       op
 *         |                       |          |
 * less ---|-----------------------|--+       |
 *         |                +---+  |  |       /2
 *    a ---|-------------+--|AND|__|__|__     |
 *         |          +--|--|and|  |  |  |    |
 *         |          |  |  +---+  |  |  |  +-+-
 *         |          |  |  +---+  |  |  +--|i0  \
 *         |          |  +--|OR |__|__|_____|i1   \
```

```
 *         |          +--|--| or|  |  |     |  mux +- res
 *         |          |  |  +---+  |  |  +--|i2   /
 *         |          |  |    +----+  +--|--|i3  /
 *         |          |  |    |          |  +---
 *         |          |  |  +---+        |
 *         |  +---+   |  +--| + |________|   less=i3
 *         +--|XOR|___+-----|sum|
 *    b ------|xor|    t    +---+
 *            +---+           |
 *                        carry_out
 *  binvert op | res
 *  -----------+------
 *        0 00 | a and b
 *        0 01 | a  or b
 *        0 10 | a add b
 *        0 11 | less
 *        1 00 | a and (not b)
 *        1 01 | a  or (not b)
 *        1 10 | a sub b (si carry_in=1)
 *        1 11 | less
 */

class ALU_BIT {

  WIRE t,                 // Salida de la puerta XOR
    i[4];                 // Entradas del multiplexor
  XOR xor;
  OR or;
  AND and;
  ADDER_BIT sum;
  MUX_4_1 mux;

public:
  void run(WIRE &a, WIRE &b,     // Bits de datos
           WIRE &binvert,        // Complemento de la l'inea "b"
           WIRE &carry_in,       // Acarreo de entrada
           WIRE op[2],           // Operaci'on de la ALU
           WIRE &less,           // Entrada directa a la salida
           WIRE &res,            // Resultado de la ALU
           WIRE &carry_out);     // Acarreo de salida
};

/*
 * ALU-MSb de 1 bit.
 *
 *      binvert               carry_in       op
 *         |                       |          |
 * less ---|-----------------------|--+       |
 *         |                +---+  |  |       /2
 *    a ---|-------------+--|AND|__|__|__     |
 *         |          +--|--|and|  |  |  |    |
 *         |          |  |  +---+  |  |  |  +-+-
 *         |          |  |  +---+  |  |  +--|i0  \
 *         |          |  +--|OR |__|__|_____|i1   \
 *         |          +--|--| or|  |  |     |  mux +-------- res
 *         |          |  |  +---+  |  |  +--|i2   /
 *         |          |  |    +----+  +--|--|i3  /
 *         |          |  |    |    |     |  +---
 *         |          |  |  +---+  |     |        +---+
 *         |  +---+   |  +--| + |__|_____+--------|XOR|_____ set
 *         +--|XOR|___+-----|sum|  |           +--| x1|
 *    b ------|xor|    t    +---+  |    +---+  |  +---+
 *            +---+           |    +----|XOR|__|____________ overflow
 *                            +---------| x0|
 *                            |         +---+
 *                        carry_out
 */

class ALU_MSBIT {
  WIRE t,                 // Salida de la puerta XOR
    i[4];                 // Entradas del multiplexor
```

```
  XOR xor;
  OR or;
  AND and;
  ADDER_BIT sum;
  MUX_4_1 mux;
  XOR x0,x1;

public:
  void run(WIRE &a, WIRE &b,
           WIRE &binvert,
           WIRE &carry_in,
           WIRE op[2],
           WIRE &less,
           WIRE &res,
           WIRE &carry_out,
           WIRE &set,
           WIRE &overflow);
};

/*
 * ALU de 32 bits ripple-carry.
 *
 *      bnegate                                       op[2]
 *         |                                            |
 *         +------+---------------+            +--------+
 *         |      |               |            |        |
 *         | +----+---------------+------------+------+ |
 *         | | binvert       carry_in         op      | |
 *         | |                                        | |
 *       +---+ less                                   | |
 *       | | |                 ALU[0]                 | |
 *  a[0]-----+ a                                  res +----+------ res[0]
 *       | | |                                        | |  |
 *  b[0]-----+ b                                      | |  |
 *       | | |                carry_out               | |  |
 *       | | +--------------------+-------------------+ |  |
 *       | |                      |                     |  |
 *       | +------+               | c[0]       +--------+  |
 *       | |      |               |            |        |  |
 *       | | +----+---------------+------------+------+ |  |
 *       | | | binvert       carry_in         op      | |  |
 *       | | |                                        | |  |
 *    0 -----+ less                                   | |  |
 *       | | |                 ALU[1]                 | |  |
 *  a[1]-----+ a                                  res +-----+----- res[1]
 *       | | |                                        | |  ||
 *  b[1]-----+ b                                      | |  ||
 *       | | |                carry_out               | |  ||
 *       | | +--------------------+-------------------+ |  ||
 *       : :                      : c[1]                :  ::
 *       | +------+               |            +--------+  ||
 *       |        |               | c[30]      |           ||
 *       |   +----+---------------+------------+------+    ||
 *       |   | binvert       carry_in         op      |    ||
 *       |   |                                        |    ||
 *    0 -----+ less                               set +-+  ||
 *       |   |                 ALU[31]                | |  ||
 * a[31]-----+ a                                  res +--------+-- res[31]
 *       |   |                                        | |  ||..|
 * b[31]-----+ b                             overflow +----------- overflow
 *       |   |                carry_out               | |  ||..|
 *       |   +--------------------+-------------------+ |  ||..|
 *       |less                    |                     |  ||..|
 *       +----------------------------------------------+  ||..|
 *                                |                        ||..|
 *                            carry_out                  +-++..+-+
 *                                                       |  NOR  |
 *                                                       |  zero |
 *                                                       +---+---+
 *                                                           |
 *                                                         zero
 *        bnegate oper | output
```

```
 *       -------------+----------
 *           * 0   00 | A and B
 *           * 0   01 | A or B
 *           * 0   10 | A add B
 *             0   11 | 000...0A donde A = overflow XOR sum[MSb]
 *             1   00 | A and (not B)
 *             1   01 | A or (not B)
 *           * 1   10 | A sub B = A add ((not B) + 1)
 *           * 1   11 | (A<B)
 *
 *           * -> Operaciones normalmente 'utiles
 */

class ALU_32BIT_RC {
  WIRE c[31];                // Bits de acarreo entre ALU's de 1 bit
  WIRE less;                 // Bit de realimentaci'on del MSb al LSb
  ALU_BIT alu[31];           // ALU's menos significativas
  ALU_MSBIT alu31;           // La ALU m'as significativa
  NOR nor;                   // C'alculo del zero
 public:
  void run(WIRE a[32], WIRE b[32],    // Buses de datos de entrada
       WIRE &bnegate, WIRE op[2], // Entradas de control de operaci'on
           WIRE res[32],              // Bus de datos de salida
           WIRE &overflow,            // Overflow en la operaci'on
           WIRE &carry_out,           // Acarreo de salida de la ALU
           WIRE &zero);               // El resultado es cero
};
```

Método:

```
#include "defs.h"
#include "gates.h"
#include "decods.h"
#include "muxs.h"
#include "alu32.h"

void ADDER_BIT::run(WIRE &a, WIRE &b, WIRE &carry_in, WIRE &sum, WIRE
&carry_out) {
  a0.run(a,carry_in,t[0]);
  a1.run(b,carry_in,t[1]);
  a2.run(a,b,t[2]);
  o.run(carry_out,3,t);
  x0.run(a,b,t[3]);
  x1.run(t[3],carry_in,sum);
}

void ALU_BIT::run(WIRE &a, WIRE &b,
                  WIRE &binvert, WIRE &carry_in, WIRE op[2],
                  WIRE &less, WIRE &res, WIRE &carry_out) {
  xor.run(binvert,b,t);
  and.run(a,t,i[0]);
  or.run(a,t,i[1]);
  sum.run(a,t,carry_in,i[2],carry_out);
  i[3]=less;
  mux.run(i,op,res);
}

void ALU_MSBIT::run(WIRE &a, WIRE &b,
                    WIRE &binvert,
                    WIRE &carry_in,
                    WIRE op[2],
                    WIRE &less,
                    WIRE &res,
                    WIRE &carry_out,
```

```
                    WIRE &set,
                    WIRE &overflow) {
  xor.run(binvert,b,t);
  and.run(a,t,i[0]);
  or.run(a,t,i[1]);
  sum.run(a,t,carry_in,i[2],carry_out);
  i[3]=less;
  mux.run(i,op,res);
  x0.run(carry_in,carry_out,overflow);
  x1.run(i[2],overflow,set);
}

void ALU_32BIT_RC::run(WIRE a[32], WIRE b[32],    // Buses de datos de
entrada
      WIRE &bnegate, WIRE op[2], // Entradas de control de operaci'on
      WIRE res[32],              // Bus de datos de salida
      WIRE &carry_out,           // Acarreo de salida de la ALU
      WIRE &overflow,            // Overflow en la operaci'on
      WIRE &zero) {              // El resultado es cero
      WIRE z;
  z=0;
  alu[0].run(a[0],b[0],bnegate,bnegate,op,less,res[0],c[0]);
  for(int j=1;j<31;j++) {
    alu[j].run(a[j],b[j],bnegate,c[j-1],op,z,res[j],c[j]);
  }

      alu31.run(a[31],b[31],bnegate,c[30],op,z,res[31],carry_out,less,
      overflow);
      nor.run(zero,32,res);

}
```

- **Un fichero de registros de 32x32:**

  Clase:

```
/*
 * Celda b'asica del fichero de registros, basada en un flip-flop D.
 *
 *                       sel0 sel1                  |     |
 *                         |    |                  +----v-----v---+
 *            +------+  +--|---BUF-- out1      |  sel0  sel1  |
 *      in ---+ D  Q +--+  |                 -->in        out1+->
 *            |      |  +-BUF------- out0      |    CELDA     |
 *   write --O> ck   |                       -O>write     out0+->
 *            +------+                         |    BASICA    |
 *                                             +--------------+
 */
class FR_BIT {
  WIRE q,qn;     // Salidas del flip-flop (aunque s'olo "q" se
utiliza)
  NEG_D_FF f;    // El flip-flop
  BUFFER_Z b[2]; // Los buffers triestado
 public:
  void run(WIRE &in,                // Entrada de datos
           WIRE &sel0, WIRE &sel1,  // Selectores
           WIRE &write,             // Orden de escritura sobre el
flip-flop
           WIRE &out0, WIRE &out1); // Salidas de la celda
};

/*
 * Celda b'asica del registro 0.
 *
```

```
 *                      sel0 sel1                  |     |
 *                         |    |                +----v-----v---+
 *           +------+  +--|---BUF-- out1         |  sel0  sel1  |
 *     in ---+ D  Q +--+  |                    -->in        out1+->
 *           |      |  +-BUF------- out0         |    CELDA     |
 *   write --O> ck  |                          -O>write     out0+->
 *           +------+                            |    BASICA    |
 *                                               +--------------+
 */
class FR0_BIT {
  BUFFER_Z b[2]; // Los buffers triestado
 public:
  void run(WIRE &in,                // Entrada de datos
      WIRE &sel0, WIRE &sel1,  // Selectores
      WIRE &write,             // Orden de escritura sobre el flip-flop
      WIRE &out0, WIRE &out1); // Salidas de la celda
};

/*
 * Fichero de registros de 32x32 bits.
 *
 *                 write in[0]            in[1]                 in[31]
 *                    |   |       s0[0]     |                      |
 *                  +---|---|-------+---------|-------+--------...--|-------+
 *           /|  /    +-|---|----+--|---------|----+--|--------...--|----+  |
 *         / 0+--+    /  |   | +--+--+--+       | +--+--+--+    s1[0] | +--+--+--+
 *      5 |  1+-         |   | | Reg 0  +----+  | | Reg 0  +----+      | | Reg 0  +----+
 *     -/-+  :| \  /     |  +-+ Bit 0  |    |  +-+ Bit 1  |    |     +-+ Bit 31 |    |
 * addr0 |  :|  \        |   | |        +-+ |  | |        +-+ |     | |        +-+ |
 *         \31+-  X     +-A | +----+---+ | |  | +----+---+ | |     | +----+---+ | |
 *           \| \  \     | N-|------+------|-|--|------+------|-|-...--|------+      | |
 *               /  \ +|-D |wck[0] s0[1]| |  |              | |s1[1] |             | |
 *               \    +-|---|-------+----|-|--|-------+----|-|-...--|-------+     | |
 *           /| /  +-/-|---|----+--|----|-|--|----+--|----|-|-...--|----+  |     | |
 *         / 0+-  X    |   | +--+--+--+ | |  | +--+--+--+ | |      | +--+--+--+ | |
 *      5 |  1+-+   /  |   | | Reg 1  +-|-+  | | Reg 1  +-|-+      | | Reg 1  +-|-+
 *     -/-+  :|    \   |  +-+ Bit 0  | | |  +-+ Bit 1  | | |     +-+ Bit 31 | | |
 * addr1 |  :|    /    |   | |        +-+ |  | |        +-+ |     | |        +-+ |
 *         \31+-+   \  +-A | +----+---+ | |  | +----+---+ | |     | +----+---+ | |
 *           \|  \/    | N-|------+------|-|--|------+------|-|-...--|------+      | |
 *                \ +\-|-D |wck[1]      | |  |              | |      |             | |
 *           w[0]/ X   :   :     s0[31] : :  :              : :      :             : :
 *                / \ +|---|-------+----|-|--|-------+----|-|-...--|-------+     | |
 *           /| //   +-|---|----+--|----|-|--|----+--|----|-|-...--|----+  |     | |
 *         / 0+-/      |   | +--+--+--+ | |  | +--+--+--+ | |s1[31]| +--+--+--+ | |
 *      5 |  1+-w[1]   |   | | Reg 31 +-|-+  | | Reg 31 +-|-+      | | Reg 31 +-|-+
 *     -/-+  :|        |  +-+ Bit 0  | | |  +-+ Bit 1  | | |     +-+ Bit 31 | | |
 * addr2 |  :|        |    |        +-+ |    |        +-+ |       |        +-+ |
 *         \31+-+      +-A  +----+---+ | |    +----+---+ | |       +----+---+ | |
 *           \|   \      N--------+------|-|---------+------|-|-...---------+      | |
 *                 +----D wck[31]       | |                 | |                    | |
 *               w[31]           out0[0] out1[0]   out0[1] out1[1]       out0[31]
 *                                                                      out1[31]
 *                     |       |
 *                  +----+-------+------+
 *                5 |    ck    write    |
 *              -/-> addr0              | 32
 *              -/-> addr1        out0 +-/->
 *              -/-> addr2              | 32
 *              32 |              out1 +-/->
 *              -/-> in    FICHERO      |
 *                 |       REGISTROS    |
 *                 +--------------------+
 */
class FR_32x32 {
  FR0_BIT bit0[32];     // Registro 0
  FR_BIT bit[31][32];   // Array de celdas b'asicas
  DECOD_5_32 dec0;      // Decodificador 0 de direcci'on para lectura
  DECOD_5_32 dec1;      // Decodificador 1 de direcci'on para lectura
  DECOD_5_32 decw;      // Decodificador de direcci'on para escritura
  AND and[32];          // Puertas de habilitaci'on de escritura
  WIRE
```

```
    s0[32],            // Salidas del primer decodificador de lectura
    s1[32],            // Salidas del segundo decodificador de lectura
    w[32],             // Salidas del decodificador de escritura
    wck[32];           // Salidas de las puertas de habilitaci'on de
escritura

 public:
  void run(WIRE in[32], WIRE addr0[5], WIRE addr1[5], WIRE addr2[5],
           WIRE &write, WIRE out0[32], WIRE out1[32]);
};
```

Método:

```
#include "defs.h"
#include "gates.h"
#include "decods.h"
#include "latches.h"
#include "flip-flops.h"
#include "fr32x32.h"

void FR_BIT::run(WIRE &in, WIRE &sel0, WIRE &sel1, WIRE &write, WIRE
&out0, WIRE &out1) {
  f.run(in,write,q,qn);
  b[0].run(q,sel0,out0);
  b[1].run(q,sel1,out1);
}

void FR0_BIT::run(WIRE &in, WIRE &sel0, WIRE &sel1, WIRE &write, WIRE
&out0, WIRE &out1) {
  WIRE gnd=GND;
  b[0].run(gnd,sel0,out0);
  b[1].run(gnd,sel1,out1);
}

void FR_32x32::run(WIRE in[32], WIRE addr0[5], WIRE addr1[5], WIRE
addr2[5],
                   WIRE &write, WIRE out0[32], WIRE out1[32]) {
  WIRE gnd=GND;
  int i;
  dec0.run(addr0,s0);
  dec1.run(addr1,s1);
  decw.run(addr2,w);
  for(i=0;i<32;i++) {
    and[i].run(w[i],write,wck[i]);
  }
  int j;
  // El registro 0 siempre contiene 0.
  // Para conseguir esto, hacemos que su entrada siempre valga 0.

  for(j=0;j<32;j++) {
    bit0[j].run(in[i],s0[0],s1[0],wck[0],out0[j],out1[j]);
  }
  for(i=1;i<32;i++) {
    for(j=0;j<32;j++) {
      bit[i-1][j].run(in[j],s0[i],s1[i],wck[i],out0[j],out1[j]);
    }
  }
}
```

- **Una memoria sram de 64x32 bits:**

Clase:

```
/*
 * Celda b'asica de memoria, basada en un cerrojo SR as'incrono.
 * Digital Logic and Computer Design. M. Morris Mano, pag 295.
 *
 * Responde a la tabla de verdad:
 *
 *  select write | Qn+1 out
 *  -------------+---------
 *       0     0 |   Qn   Z
 *       0     1 |   Qn   Z
 *       1     0 |   Qn  Qn
 *       1     1 |   in  in
 *
 * y su implementaci'on f'isica es:
 *
 *              write
 *        +---+   |   +-----+
 *        |N  |nin| +-+A    |   +------+
 *      +-+O n+---|-|-+N a0+---+R   Qn+- qn
 *      | |T  |   +-|-+D    | r | FFSR |
 *  in --+ +---+   | | +-----+   |  Nor |
 *      |          | | +-----+   |      |
 *      |          | +-+A    |   |   f  |  q +---------+
 *      +----------|-|-+N a1+---+S    Q+----+BUFFER z+- out
 *                 +-|-+D    | s +------+    +----+---+
 *                   | +-----+                     |
 *                   +-------------+-------------+
 *                                 |
 *                               select
 *
 * que en lo sucesivo ser'a:
 *
 *            write
 *              |
 *        +-----+-----+
 *        |           |
 *        |           |
 *    in --+ SRAM_BIT  +-- put
 *        |           |
 *        |           |
 *        +-----+-----+
 *              |
 *            select
 *
 */
class SRAM_BIT {
  NOT n;
  WIRE nin,r,s,q,qn;
  AND a0,a1;
  SR_LATCH_NOR f;
  BUFFER_Z b;
 public:
  void run(WIRE &in, WIRE &select, WIRE &write, WIRE &out);
};
/*
 * Este es el dise~no de un bloque de 4x4 bits de memoria SRAM.
 * Se utilizar'a m'as adelante en la construcci'on de la memoria
 * SRAM mediante replicaci'on.
 *
 *   blksel |     |in[3]      |in[2]      |in[1]      |in[0]
 *          |     |           |           |           |         write
 *  <-------|-----|----+------|----+------|----+------|----+------+--
 *          |     |    |      |    |      |    |      |    |      |
 *          |     |  +-v-+    |  +-v-+    |  +-v-+    |  +-v-+    |
 *          |     +--> b +--+ +--> b +--+ +--> b +--+ +--> b +--+ |
 *          |     |  +-+-+  | |  +-+-+  | |  +-+-+  | |  +-+-+  | |
 *         +-A    |    |s[0]| |    |    | |    |    | |    |    | |
 *       +-|-N--|----+----|-|----+----|-|----+----|-|----+      | |
 *  sel[0]|  | D   |          | |         | |         | |         | |
 *  ------+-|-----|----------|-|---------|-|---------|-|---------|-|--->
 *          |     |    +-----|-|----+----|-|----+----|-|----+----|-+
 *          |     |    |     | |    |    | |    |    | |    |    | |
```

```
 *          |    |  +-v-+  | |  +-v-+  | |  +-v-+  | |  +-v-+  | |
 *          |    +--> b +--+ +--> b +--+ +--> b +--+ +--> b +--+ |
 *          |    |  +-+-+  | |  +-+-+  | |  +-+-+  | |  +-+-+  | |
 *          +-A  |    |s[1]| |    |    | |    |    | |    |    | |
 *        +-|-N--|----+----|-|----+----|-|----+----|-|----+    | |
 *  sel[1]| | D  |         | |         | |         | |         | |
 *  ------+-|----|---------|-|---------|-|---------|-|---------|-|--->
 *          |    |    +----|-|----+----|-|----+----|-|----+----|-+
 *          |    |    |    | |    |    | |    |    | |    |    | |
 *          |    |  +-v-+  | |  +-v-+  | |  +-v-+  | |  +-v-+  | |
 *          |    +--> b +--+ +--> b +--+ +--> b +--+ +--> b +--+ |
 *          |    |  +-+-+  | |  +-+-+  | |  +-+-+  | |  +-+-+  | |
 *          +-A  |    |s[2]| |    |    | |    |    | |    |    | |
 *        +-|-N--|----+----|-|----+----|-|----+----|-|----+    | |
 *  sel[2]| | D  |         | |         | |         | |         | |
 *  ------+-|----|---------|-|---------|-|---------|-|---------|-|--->
 *          |    |    +----|-|----+----|-|----+----|-|----+----|-+
 *          |    |    |    | |    |    | |    |    | |    |    |
 *          |    |  +-v-+  | |  +-v-+  | |  +-v-+  | |  +-v-+  |
 *          |    +--> b +--+ +--> b +--+ +--> b +--+ +--> b +--+
 *          |       +-+-+  |     +-+-+  |     +-+-+  |     +-+-+  |
 *          +-A     |s[3]|      |    |      |    |      |    |
 *        +---N-------+----|------+----|------+----|------+    |
 *  sel[3]|   D            |           |           |           |
 *  ------+----------------|-----------|-----------|-----------|----->
 *                         v           v           v           v
 *                       out[3]|     out[2]|     out[1]|     out[0]|
 *
 * En adelante, a este dispositivo lo referenciaremos como:
 *
 *        selblk in[3-0]
 *            |  | | | |
 *          +-+--+-+-+-+-+
 *          |            +-- write
 * sel[0] --+            |-- sel[0] (son las mismas entradas)
 * sel[1] --+    4x4     |-- sel[1]
 * sel[2] --+    SRAM    |-- sel[2]
 * sel[3] --+            |-- sel[3]
 *          +----+-+-+-+-+
 *               | | | |
 *              out[3-0]
 *
 */
class BIT_4x4_Array {
  SRAM_BIT b[4][4];
  AND a[4];
  WIRE s[4];
 public:
  void run(WIRE in[4], WIRE &blksel, WIRE sel[4], WIRE &write, WIRE
out[4]);
};

/*
 * Memoria SRAM de 16x4 bits, usando 4 bloques de 4x4 bits.
 *
 * select------>/| blksel[]                                      in[3]
 *             |0+----------------------------------------------+ |in[2]
 *           __|1+--------------------------------+             | | |in[1]
 *          |  |2+------------------+             |             | | | |in[0]
 * addr[3-2]|  |3+----+             |             |             | | | | |
 *          |   \|    | +-----------|-+-----------|-+-----------|-+ | | |
 *          |         | | +---------|-|-+---------|-|-+---------|-|-+ | |
 *          |         | | | +-------|-|-|-+-------|-|-|-+-------|-|-|-+ |
 *          |         | | | | +-----|-|-|-|-+-----|-|-|-|-+-----|-|-|-|-+   w
 * addr[3-0]|         | | | | |     | | | | |     | | | | |     | | | | |   r
 * ---------+   /|sel +-+-+-+-++<--++-+-+-+-++<--++-+-+-+-++<--++-+-+-+-++<-i
 *          |  |0+--->         |--->         |--->         |--->         |  t
 * addr[1-0]|__|1+--->   4x4   |--->   4x4   |--->   4x4   |--->   4x4   |  e
 *             |2+--->   SRAM  |--->   SRAM  |--->   SRAM  |--->   SRAM  |
 *             |3+--->         |--->         |--->         |--->         |
 *              \|   +--+-+-+-++   +--+-+-+-++   +--+-+-+-++   +--+-+-+-++
 *                      | | | |       | | | |       | | | |       | | | |
 *                      +-|-|-|-------+-|-|-|-------+-|-|-|-------+ | | |
 *                        +-|-|---------+-|-|---------+-|-|-------|-+ | |
 *                          +-|-----------+-|-----------+-|-------|-|-+ |
```

```
 *                             +-------------+-------------+-------|-|-|-+
 *                                                                 | | | |
 *                                                                 | | |out[0]
 *                                                                 | |out[1]
 *                                                                 |out[2]
 *                                                                out[3]
 * En adelante, el bloque ser'a:
 *
 *                 in[3-0]
 *                     |
 *             +-------+-------+
 *             |               |
 *    select --+               |
 *             |     16x4      +-- write
 *             |     SRAM      |
 * addr[3-0] --+               |
 *             |               |
 *             +-------+-------+
 *                     |
 *                  out[3-0]
 */
class SRAM_16x4 {
  BIT_4x4_Array blk[4];
  DECOD_2_4 dec;
  DECOD_2_4_SEL dec_sel;
  WIRE sel[4],blksel[4];

 public:
  void run(WIRE &select, WIRE &write, WIRE addr[4], WIRE in[4], WIRE
out[4]);
};

/*
 * Memoria SRAM de 16x32 bits.
 *
 * Se trata de 32/4=8 bloques de 16x4 bits, direccionados en paralelo, pero que
 * toman un dan bits de entrada y salida distintos.
 *
 *            in[31-28] in[27-24]    ...   in[3-0]
 *                 |         |                 |
 *             +---+---+ +---+---+ +-...-+ +---+---+
 *    select --+       |-+       |-+     |-+       |
 *             | 16x4  +-| 16x4  +-|     +-| 16x4  +-- write
 * addr[3-0] --+       |-+       |-+     |-+       |
 *             +-------+ +-------+ +-...-+ +-------+
 *                 |         |                 |
 *           out[31-28] out[27-24]   ...  out[3-0]
 *
 * Dispositivo que en adelante ser'a:
 *
 *                 in[31-0]
 *                     |
 *             +-------+-------+
 *             |               |
 *    select --+               |
 *             |     16x32     +-- write
 *             |     SRAM      |
 * addr[3-0] --+               |
 *             |               |
 *             +-------+-------+
 *                     |
 *                 out[31-0]
 */

class SRAM_16x32 {
  SRAM_16x4 blk[8];

 public:
  void run(WIRE &select, WIRE &write, WIRE addr[4], WIRE in[32], WIRE
out[32]);
};
```

```
/*
 * Memoria SRAM de 64x32 bits
 *
 * addr[5,4] --------+     in[31-0]
 *                   |        |
 *                +--+--+     |
 * select -------/0 1 2 3\    +-------+
 *              +-+-+-+-+-+   |       |
 *                | | | |     |  +----+----------+
 *                | | | |     |  |               |
 *                | | | +-----|--+ select        |
 *                | | |       |  |     blk[3]    +-----+------- write
 * addr[3-0] ---+-|-|-|-------|--+     16x32     |     |
 *              | | | |       |  |     SRAM      |     |
 *              | | | |       |  |               |     |
 *              | | | |       |  +----------+----+     |
 *              | | | |       |             |          |
 *              | | | |       +-------+     +-------+  |
 *              | | | |       |       |             |  |
 *              | | | |       |  +----+----------+  |  |
 *              | | | |       |  |               |  |  |
 *              | | | +-------|--+ select        |  |  |
 *              | | |         |  |     blk[2]    +--|--+
 *              +-|-|---------|--+     16x32     |  |  |
 *              | | |         |  |     SRAM      |  |  |
 *              | | |         |  |               |  |  |
 *              | | |         |  +----------+----+  |  |
 *              | | |         |             |       |  |
 *              | | |         +-------+     +-------+  |
 *              | | |         |       |             |  |
 *              | | |         |  +----+----------+  |  |
 *              | | |         |  |               |  |  |
 *              | | +---------|--+ select        |  |  |
 *              | |           |  |     blk[1]    +--|--+
 *              +-|-----------|--+     16x32     |  |  |
 *              | |           |  |     SRAM      |  |  |
 *              | |           |  |               |  |  |
 *              | |           |  +----------+----+  |  |
 *              | |           |             |       |  |
 *              | |           +-------+     +-------+  |
 *              | |                   |             |  |
 *              | |               +----+----------+  |  |
 *              | |               |               |  |  |
 *              | +--------------+ select        |  |  |
 *              |                |     blk[0]    +--|--+
 *              +----------------+     16x32     |  |
 *                               |     SRAM      |  |
 *                               |               |  |
 *                               +----------+----+  |
 *                                          |       |
 *                                          +-------+
 *                                                  |
 *                                             out[31,0]
 *
 */
class SRAM_64x32 {
  SRAM_16x32 blk[4];
  DECOD_2_4_SEL dec;
  WIRE select[4];
 public:
  void run(WIRE &select, WIRE &write, WIRE addr[6], WIRE in[32], WIRE
out[32]);
};
```

Método:

```
#include "defs.h"
#include "gates.h"
#include "decods.h"
#include "muxs.h"
#include "latches.h"
#include "srams.h"
```

```
void SRAM_BIT::run(WIRE &in, WIRE &select, WIRE &write, WIRE &out) {
  n.run(in,nin);
  a0.run(r,3,write,nin,select);
  a1.run(s,3,write,in,select);
  f.run(s,r,q,qn);
  b.run(q,select,out);
}

void BIT_4x4_Array::run(WIRE in[4], WIRE &blksel,
                        WIRE sel[4], WIRE &write, WIRE out[4]) {
  int i,r,c;
  for(i=0;i<4;i++) a[i].run(blksel,sel[i],s[i]);
  for(r=0;r<4;r++)
    for(c=0;c<4;c++)
      b[r][c].run(in[c],s[r],write,out[c]);
}

void SRAM_16x4::run(WIRE &select, WIRE &write, WIRE addr[4],
                    WIRE in[4], WIRE out[4]) {
  dec_sel.run(addr+2,select,blksel);
  dec.run(addr,sel);
  for(int i=0;i<4;i++) {
    blk[i].run(in,blksel[i],sel,write,out);
  }
}

void SRAM_16x32::run(WIRE &select, WIRE &write, WIRE addr[4],
                     WIRE in[32], WIRE out[32]) {
  for(int i=0;i<8;i++) {
    blk[i].run(select,write,addr,in+i*4,out+i*4);
  }
}

void SRAM_64x32::run(WIRE &select, WIRE &write, WIRE addr[6],
                     WIRE in[32], WIRE out[32]) {
  dec.run(addr+4,select,this->select);
  for(int i=0;i<4;i++) {
    blk[i].run(this->select[i],write,addr,in,out);
  }
}
```

- **Una unidad de control de multiciclo:**

  Clase:

```
class CONTROL{

  WIRE o[18],nin[10],NS[4],S[4];
  NOT     not[10];
  AND     and[18];
  OR      or[21];
  REG_4   reg;

 public:
  CONTROL();
  void run (WIRE &ck, WIRE in[6], WIRE out[17]);

};
```

  Método:

```
#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <stdlib.h>
#include "defs.h"
#include "gates.h"
#include "clock.h"
#include "decods.h"
#include "muxs.h"
#include "latches.h"
#include "flip-flops.h"
#include "regs.h"
#include "control.h"

CONTROL::CONTROL()
{
  int i;
  WIRE cero_1_[4] = {0,0,0,0},
                out_1_[4],
                ck = VCC;
  for(i=0;i<20;i++) reg.run(ck,cero_1_,out_1_);
  ck = GND;
  for(i=0;i<20;i++) reg.run(ck,cero_1_,out_1_);

}

void CONTROL::run(WIRE &ck,WIRE in[6], WIRE out[17]) {

  int i;

  not[0].run(S[0],nin[0]);
  not[1].run(S[1],nin[1]);
  not[2].run(S[2],nin[2]);
  not[3].run(S[3],nin[3]);

  not[4].run(in[0],nin[4]);
  not[5].run(in[1],nin[5]);
  not[6].run(in[2],nin[6]);
  not[7].run(in[3],nin[7]);
  not[8].run(in[4],nin[8]);
  not[9].run(in[5],nin[9]);

  and[0].run(o[0],4,nin[0],nin[1],nin[2],nin[3]);
  and[1].run(o[1],4,S[0],nin[1],nin[2],nin[3]);
  and[2].run(o[2],4,nin[0],S[1],nin[2],nin[3]);
  and[3].run(o[3],4,S[0],S[1],nin[2],nin[3]);
  and[4].run(o[4],4,nin[0],nin[1],S[2],nin[3]);
  and[5].run(o[5],4,S[0],nin[1],S[2],nin[3]);
  and[6].run(o[6],4,nin[0],S[1],S[2],nin[3]);
  and[7].run(o[7],4,S[0],S[1],S[2],nin[3]);
  and[8].run(o[8],4,nin[0],nin[1],nin[2],S[3]);
  and[9].run(o[9],4,S[0],nin[1],nin[2],S[3]);

and[10].run(o[10],10,S[0],nin[1],nin[2],nin[3],nin[4],in[1],nin[6],nin
[7],nin[8],nin[9]);

and[11].run(o[11],10,S[0],nin[1],nin[2],nin[3],nin[4],nin[5],in[2],nin
[7],nin[8],nin[9]);
```

```
and[12].run(o[12],10,S[0],nin[1],nin[2],nin[3],nin[4],nin[5],nin[6],ni
n[7],nin[8],nin[9]);

and[13].run(o[13],10,nin[0],S[1],nin[2],nin[3],in[0],in[1],nin[6],in[3
],nin[8],in[5]);

and[14].run(o[14],10,S[0],nin[1],nin[2],nin[3],in[0],in[1],nin[6],nin[
7],nin[8],in[5]);

and[15].run(o[15],10,S[0],nin[1],nin[2],nin[3],in[0],in[1],nin[6],in[3
],nin[8],in[5]);

and[16].run(o[16],10,nin[0],S[1],nin[2],nin[3],in[0],in[1],nin[6],nin[
7],nin[8],in[5]);

and[17].run(o[17],10,nin[0],S[1],nin[2],nin[3],in[0],in[1],nin[6],nin[
7],nin[8],in[5]);

  or[0].run(NS[0],6,o[0],o[6],o[10],o[13],o[16],o[17]);   /*NS0*/
  or[1].run(NS[1],5,o[6],o[12],o[14],o[15],o[16]);   /*NS1*/
  or[2].run(NS[2],4,o[3],o[6],o[12],o[13]);   /*NS2*/
  or[3].run(NS[3],2,o[10],o[11]);   /*NS3*/
  or[4].run(out[0],1,o[7]);   /*RegDst*/
  or[5].run(out[1],2,o[4],o[7]);   /*RegWrite*/
  or[6].run(out[2],7,o[2],o[3],o[4],o[5],o[6],o[7],o[8]);  /*AluselA*/
  or[7].run(out[3],2,o[0],o[1]);   /*AluselB0*/
  or[8].run(out[4],5,o[1],o[2],o[3],o[4],o[5]);    /*AluselB1*/
  or[9].run(out[5],1,o[8]);   /*Aluop0*/
  or[10].run(out[6],2,o[6],o[7]);     /*Aluop1*/
  or[11].run(out[7],1,o[1]);   /*Target Write*/
  or[12].run(out[8],1,o[8]);   /*PcSource0*/
  or[13].run(out[9],1,o[9]);   /*PcSource1*/
  or[14].run(out[10],1,o[4]);   /*Memtoreg*/
  or[15].run(out[11],1,o[0]);   /*IRWrite*/
  or[16].run(out[12],1,o[5]);   /*MemWrite*/
  or[17].run(out[13],3,o[0],o[3],o[4]);   /*MemRead*/
  or[18].run(out[14],3,o[3],o[4],o[5]);   /*IorD*/
  or[19].run(out[15],1,o[8]);   /*PcWriteCond*/
  or[20].run(out[16],2,o[0],o[9]);     /*PcWrite*/

  reg.run(ck,NS,S);

}
```

- **Una unidad de control para la alu:**

  Clase:

```
class CONTROL_ALU{
  OR or[3];
  AND and[2];
  NOT not[2];
  WIRE naluop,nfunct,o,a;
 public:
  void run(WIRE funct[6],WIRE aluop[2],WIRE operacion[3]);
};
```

  Método:

```
#include "defs.h"
```

```
#include "gates.h"
#include "decods.h"
#include "muxs.h"
#include "ctrlalu.h"

void CONTROL_ALU :: run(WIRE funct[6],WIRE aluop[2],WIRE
operacion[3]){

  not[0].run(aluop[1],naluop);
  not[1].run(funct[2],nfunct);
  or[0].run(funct[0],funct[3],o);
  or[1].run(naluop,nfunct,operacion[1]);
  and[0].run(aluop[1],funct[1],a);
  and[1].run(aluop[1],o,operacion[0]);
  or[2].run(aluop[0],a,operacion[2]);

}
```

- **Una unidad de sigo extendido**:

  Clase:

```
class SIGNO_EXTENDIDO {

public:

      void run(WIRE in[16],WIRE out[32]);

};
```

  Método:

```
/***Este m,todo extiende en signo 16 bits menos significativos***/
#include "defs.h"
#include "extend.h"

void SIGNO_EXTENDIDO::run (WIRE in[16],WIRE out[32])
{
  int i;
  if(in[15]==GND)
    {
      for (i=31;i>=16;i--) out[i]=GND;
      for (i=15;i>=0;i--) out[i]=in[i];
    }
  else
  {
    for (i=31;i>=16;i--) out[i]=VCC;
    for (i=15;i>=0;i--) out[i]=in[i];
  }
}
```

- **Una unidad de desplazamiento:**

  Clase:

```
class DESPLAZAMIENTO_2_IZQ {

public:

      void run(WIRE in[32],WIRE out[32]);
```

```
};

class DESPLAZAMIENTO_MAS_PC {

 public:

      void run(WIRE in[32],WIRE in_pc[32],WIRE out[32]);

};
```

Método:

```
/***Este m,todo implementa un desplazamiento a la izquierda de 2
bits***/
#include "defs.h"
#include "desplaz.h"

void DESPLAZAMIENTO_2_IZQ::run (WIRE in[32],WIRE out[32])
{
  int i;
  out[0]=GND;out[1]=GND;
  for (i=31;i>=2;i--) out[i]=in[i-2];
}
void DESPLAZAMIENTO_MAS_PC::run(WIRE in[32],WIRE in_pc[32],WIRE
out[32])
{
  int i;
  out[0]=GND;
  out[1]=GND;
  for(i=31;i>=2;i--) out[i]=in[i];
  for(i=31;i>=28;i--) out[i]=in_pc[i];
}
```

- ## El microprocesador :

Clase:

```
class MULTICICLO {

  WIRE out_signo_extend[32];

  WIRE out_desplaz1[32];
  WIRE out_desplaz2[32];

  WIRE zero,overflow,carry_out,bnegate;

  WIRE //lineas de las puertas l¢gicas
    in_and,out_and,out_or;

  WIRE  //buses de entrada/salida de 32 bits.
    in_pc[32], out_pc[32], out_ri[32], in_destino[32],
    out_destino[32], dato1[32], result[32], in_desplaz[32],
    out_desplaz[32], in_extend[16], out_extend[32];

  WIRE  //BUSES DE 5 BITS
    desplaz[15],desplaz_25_0_[26];

  WIRE //entradas del control
    out_and_pc,out_and_ri,out_and_fr,out_and_target,
    //unidad de control
```

```
    linea_control[21],
    //control alu
    alu_control[3];

  WIRE  //salidas de los multiplexores
    out_RegDst[32],out_MemtoReg[32],out_ALUSelA[32],
    out_ALUSelB[32],out_PCSource[32];

  REG_32 pc,ri, destino;

  FR_32x32  registro;

  SIGNO_EXTENDIDO signo_extendido;

  DESPLAZAMIENTO_2_IZQ desplaz1;

  DESPLAZAMIENTO_MAS_PC desplaz2;

  CONTROL control;

  CONTROL_ALU control_alu;

  MUX_2_1 mux_IorD[32],
    mux_RegDst[32],
    mux_MemtoReg[32],
    mux_ALUSelA[32];

  MUX_4_1 mux_ALUSelB[32],
    mux_PCSource[32];

  ALU_32BIT_RC alu;

  AND and,and_pc,and_ri,and_fr,and_target;

  OR or;

 public:
  MULTICICLO();
  void run (WIRE &reloj,WIRE in[32],WIRE &write,WIRE addr[32],WIRE
out[32]);
};
```

Método:

```
#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <stdlib.h>
#include "defs.h"
#include "clock.h"
#include "desplaz.h"
#include "extend.h"
#include "gates.h"
#include "decods.h"
#include "muxs.h"
#include "alu32.h"
#include "ctrlalu.h"
#include "latches.h"
#include "flip-flops.h"
#include "fr32x32.h"
#include "regs.h"
```

```
#include "counters.h"
#include "srams.h"
#include "control.h"
#include "r2000.h"

MULTICICLO::MULTICICLO() {
  int i;
  WIRE ck = VCC;
  WIRE cero[32] = {0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0,
               0,0,0,0,0,0,0,0};

  WIRE out1[32],out2[32],out3[32];

  for(i=0;i<20;i++)
    {
      pc.run(ck,cero,out1);
      destino.run(ck,cero,out2);
      ri.run(ck,cero,out3);
    }
  ck = GND;
  for(i=0;i<20;i++)
    {
      pc.run(ck,cero,out1);
      destino.run(ck,cero,out2);
      ri.run(ck,cero,out3);
    }
}

void MULTICICLO :: run( WIRE &reloj, WIRE in[32], WIRE &write, WIRE
addr[32], WIRE out[32])
{
  int i,j;
  WIRE vcc=VCC;
  WIRE
    uno[32]={255,0,0,0,0,0,0,0,
           0,0,0,0,0,0,0,0,
           0,0,0,0,0,0,0,0,
           0,0,0,0,0,0,0,0};

  /*******multiplexores************/
  for(i=0;i<32;i++)
     mux_IorD[i].run(out_pc[i],result[i],linea_control[14],addr[i]);
  for(i=0;i<32;i++)
      mux_RegDst[i].run(out_ri[i+16],out_ri[i+11],linea_control[0],out
            _RegDst[i]);
  for(i=0;i<32;i++)
            mux_MemtoReg[i].run(result[i],in[i],linea_control[10],out_M
      emtoReg[i]);
  for(i=0;i<32;i++)
      mux_ALUSelA[i].run(out_pc[i],dato1[i],linea_control[2],out_ALUSe
                        lA[i]);

  for(i=0;i<32;i++)
      mux_ALUSelB[i].run(out[i],uno[i],out_signo_extend[i],out_desplaz
                        1[i],linea_control+3,out_ALUSelB[i]);

  for(i=0;i<32;i++)
      mux_PCSource[i].run(result[i],out_destino[i],out_desplaz2[i],vcc
                                    ,linea_control+8,out_PCSource[i]);
```

```
/*******contador de programa*******/
pc.run(out_and_pc,out_PCSource,out_pc);

/******unidades funcionales**********/

ri.run(out_and_ri,in,out_ri);

registro.run(out_MemtoReg,out_ri+21,out_ri+16,out_RegDst,out_and_fr,da
        to1,out);
control.run(reloj,out_ri+26,linea_control);

write=linea_control[12];

and_pc.run(out_or,reloj,out_and_pc);

and_ri.run(linea_control[11],reloj,out_and_ri);

and_fr.run(linea_control[1],reloj,out_and_fr);

and_target.run(linea_control[7],reloj,out_and_target);

signo_extendido.run(out_ri,out_signo_extend);

desplaz1.run(out_signo_extend,out_desplaz1);

control_alu.run(out_ri,linea_control+5,alu_control);

alu.run(out_ALUSelA,out_ALUSelB,alu_control[2],alu_control,result,over
              flow,carry_out,zero);
destino.run(out_and_target,result,out_destino);

desplaz2.run(desplaz_25_0_,out_pc,out_desplaz2);

/*********puertas and y or***************/

and.run(zero,linea_control[15],out_and);

or.run(out_and,linea_control[16],out_or);

/**************SALIDA POR PANTALLA******************/

cout<< "PCWrite: " << (int)linea_control[16] << endl;

// getchar();

cout<<"Lineas_control: ";
for(i=16;i>=0;i--) cout<<"  "<<(int)linea_control[i];
cout<<endl;

cout<<"out_and_target: "<< (int)out_and_target<<endl;

cout<<"out_and_fr: "<< (int)out_and_fr<<endl;

cout<<"out_and_ri: "<< (int)out_and_ri<<endl;

cout<<"out_and_pc: "<< (int)out_and_pc<<endl;

cout<<"in: ";
for(i=31;i>=0;i--) cout<<"  "<<(int)in[i];
cout<<endl;
```

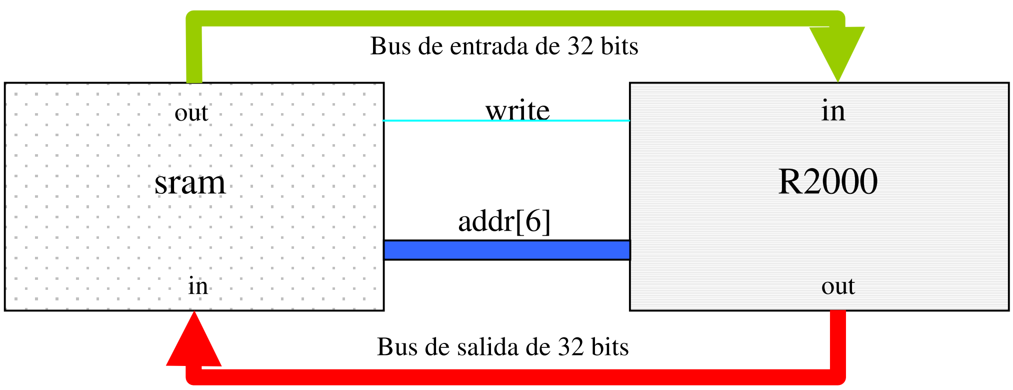
```
cout<<"IRWrite: "<<(int)linea_control[11]<<"\n";


cout<<"PCSource: "<<(int)linea_control[9]<< ' '
                  <<(int)linea_control[8]
                  <<"\n";

cout << "out_PCSource=";
for(i=31;i>=0;i--) cout <<setw(4)<< (int)out_PCSource[i];
cout  << "\n";

cout << "result= ";
for(i=31;i>=0;i--) cout <<"   "<< (int)result[i];
cout << "\n";

cout << "addr= ";
for(i=31;i>=0;i--) cout <<"   "<< (int)addr[i];
cout << "\n";

cout << "IorD= ";
cout <<"   "<< (int)linea_control[14];
cout << "\n";

cout<<"out_ALUSelA: ";
for(i=31;i>=0;i--) cout<<"  "<<(int)out_ALUSelA[i];
cout<<endl;

cout<<"ALUSelA: ";
cout<<"  "<<(int)linea_control[2];
cout<<endl;

cout<<"out_ALUSelB: ";
for(i=31;i>=0;i--) cout<<"  "<<(int)out_ALUSelB[i];
cout<<endl;

cout<<"out: ";
for(i=31;i>=0;i--) cout<<"  "<<(int)out[i];
cout<<endl;

cout<<"uno: ";
for(i=31;i>=0;i--) cout<<"  "<<(int)uno[i];
cout<<endl;

cout<<"out_signo_extend: ";
for(i=31;i>=0;i--) cout<<"  "<<(int)out_signo_extend[i];
cout<<endl;

cout<<"out_desplaz1: ";
for(i=31;i>=0;i--) cout<<"  "<<(int)out_desplaz1[i];
cout<<endl;

cout<<"ALUSelB: ";
cout<<"  "<<(int)linea_control[4]<<"  "<<(int)linea_control[3];
cout<<endl;

cout<<"out_pc: ";
for(i=31;i>=0;i--) cout<<"  "<<(int)out_pc[i];
cout<<endl;

cout<<"dato1: ";
```

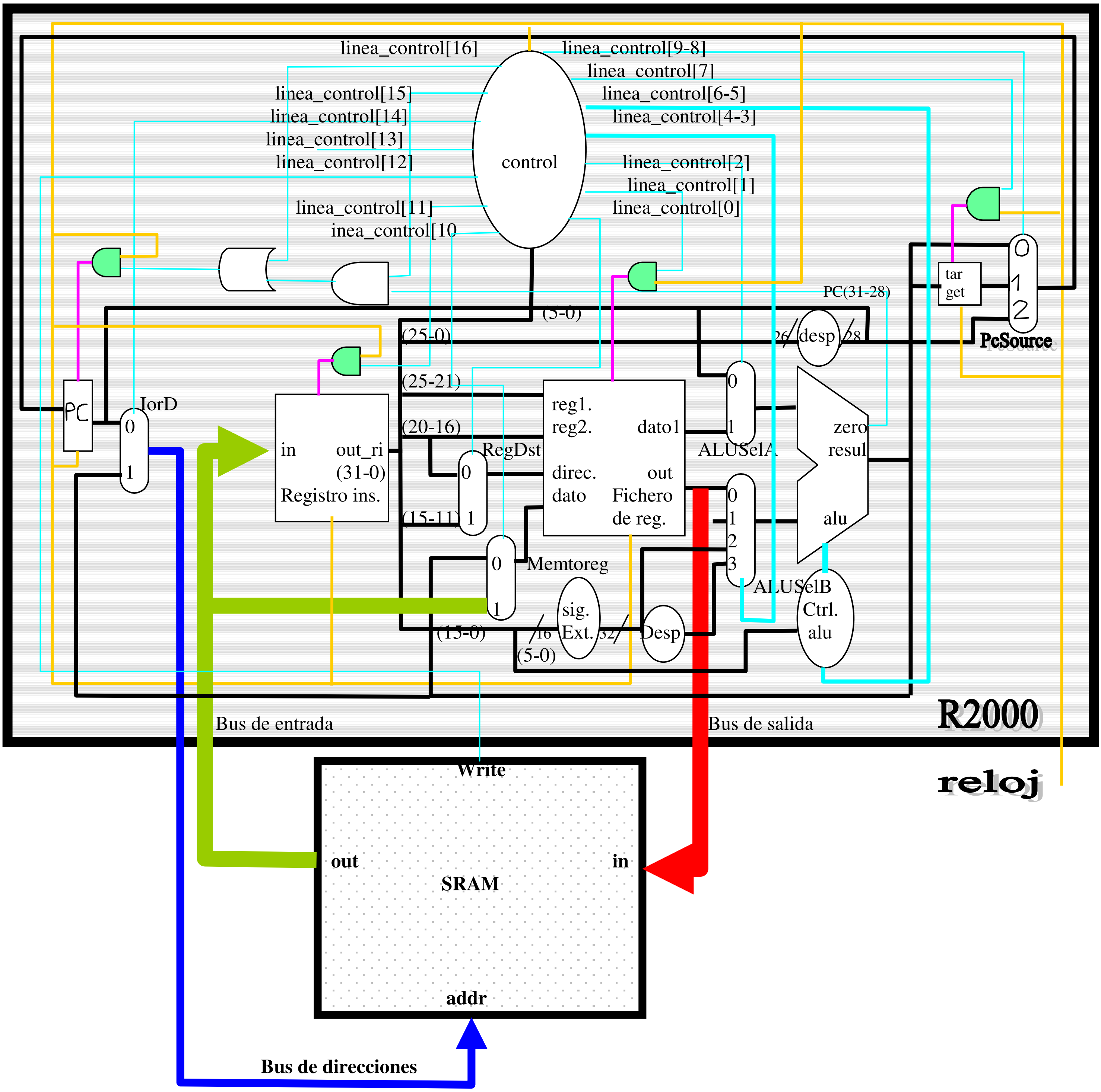
```
  for(i=31;i>=0;i--) cout<<"  "<<(int)dato1[i];
  cout<<endl;

  cout << "alu_control: "
       <<(int)alu_control[2]<<"  "
       <<(int)alu_control[1]<<"  "
       <<(int)alu_control[0]<<endl;

  cout << "ALUOp: ";
  cout << "    "<< (int)linea_control[6]<< "    "
                << (int)linea_control[5];
  cout << endl;

  cout << "TargetWrite: "<<(int)linea_control[7]<<endl;

  cout << "MemRead: ";
  cout << "    "<< (int)linea_control[13];
  cout << endl;

  cout << "RegWrite: ";
  cout << "    "<< (int)linea_control[1];
  cout << endl;

  cout << "MemtoReg: ";
  cout << "    "<< (int)linea_control[10];
  cout << endl;

  cout << "RegDst: ";
  cout << "    "<< (int)linea_control[0];
  cout << endl;

  cout << "MemWrite: ";
  cout << (int)linea_control[12] << endl;
  cout << "out_ri: ";
  for(i=31;i>=0;i--) cout<<"   "<<(int)out_ri[i];
  cout <<"\n";

  /****************FIN DE SALIDA POR PANTALLA********************/

}
```

3. Funcionamiento y dificultades en la implementación.

- **Funcionamiento.**

Para explicar el funcionamiento, lo mejor es observar las figuras siguientes:





Como podemos ver, nuestro simulador está compuesto de una serie de elementos, los cuales están declarados uno a uno como objetos, desde las puertas *or* hasta lo que es el *microprocesador*, pasando por la memoria y los buses.

El funcionamiento del simulador está compuesto por varios puntos a tener en cuenta:

- **Entrada de argumentos.**

Para ello hemos utilizado la entrada de argumentos estándar de C.

Como primer argumento se debe introducir el nombre del programa que queremos simular. Dicho programa debe estar escrito en binario en forma de matriz de 64 columnas por 32 filas, ya que esta es la longitud de la memoria que vamos a utilizar.

Como segundo argumento se introduce el nº de iteraciones de simulación.

Como tercer argumento se escribe la duración del ciclo de reloj.

- **Carga del programa en memoria.**

Se compone de dos partes:

- **Lectura del archivo:**

El archivo (primer argumento), es leido como un fichero cualquiera y su valor es almacenado en una matriz llamada *cadena.*

- **Cargado del archivo en memoria:**

Para poder escribir en memoria, primero hay que habilitarla para ello, activando la correspondiente línea de habilitación.

Seguidamente para guardar el contenido de *cadena* en las posiciones correspondientes de memoria, es necesario ir contando posiciones de 0 a 64 mediante un algoritmo de conteo en binario.

Entonces se va llamando a la memoria en cada iteración y se van guardando los datos en forma de variables útiles al simulador (VCC y GND).

- **Simulación del programa en el microprocesador.**

En este paso se ejecuta el programa durante un determinado nº de iteraciones que es introducido como segundo argumento de entrada.

La memoria atiende a las peticiones del procesador, y este le suministra un bus de entrada, una habilitación para poder escribir en ella, y un bus de direcciones.

El procesador controla la memoria, a la vez que el funcionamiento de este depende del reloj del sistema, que va cambiando según pasen un nº determinado de iteraciones de simulación (tercer argumento).

- **Presentación de los resultados.**

Para mostrar por pantalla el contenido final de la memoria después de la simulación, hemos creado un módulo que va leyendo la memoria y mostrándola por pantalla en forma de ceros y unos.

- **Codigo del programa principal.**

```
/*********************************************************************/
/*          SIMULADOR DEL PROCESADOR MIPS                          */
/*                  R2000                                        */
/*********************************************************************/
#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <stdlib.h>
#include "defs.h"
#include "clock.h"
#include "desplaz.h"
#include "extend.h"
#include "gates.h"
#include "decods.h"
```

```
#include "muxs.h"
#include "alu32.h"
#include "ctrlalu.h"
#include "latches.h"
#include "flip-flops.h"
#include "fr32x32.h"
#include "regs.h"
#include "counters.h"
#include "srams.h"
#include "control.h"
#include "r2000.h"

WIRE vcc=VCC,gnd=GND;
SRAM_64x32 memoria;
WIRE //buses de memoria-procesador
seleccion=VCC,         //habilitacion de la memoria
  MemWrite,       //señal de lectura-escritura
  out_IorD[32],    //bus de direcciones memoria
  dato2[32],      //entrada memoria,salida procesador
  reloj,          //reloj del multiciclo
  out_memoria[32];//salide memoria,entrada procesador
MULTICICLO  procesador;

char cadena[64][40];

/************PROTOTIPOS***********/
void leer_arch(char *nomfi);
void carga_memoria();
void ejecutar(int iter,int longitud);
void escribir_memoria();
/**********FIN DE PROTOTIPOS***********/
int main(int argc, char *argv[])
{
  /*********DECLARACIàN DE VARIABLES***************/
  //char nomfi[30]; //nombre del archivo con el programa en binario
  int       iter,     //numero de iteraciones de simulacion
      longitud; //longitud del ciclo de reloj
  /**** Obtencion de parametros desde la linea de comandos******/
  if( argc<3 )
  {
    fprintf(stderr,"Nombre del archivo a analizar\n");
    fprintf(stderr,"N° de iteraciones\n");
    fprintf(stderr,"Longitud del ciclo de reloj\n");
    return 1;
  }
  iter = atoi (argv[2]);
  longitud = atoi (argv[3]);
  /************FIN DECLARACIàN*****************/
  leer_arch(argv[1]);
  carga_memoria();
  ejecutar(iter,longitud);
  escribir_memoria();
  return 0;

}
/***********FIN DE LA LECTURA DEL ARCHIVO .BIN**************/

/************CARGA DE INSTRUCCIONES EN MEMORIA**************/
void carga_memoria()
{
  int i,k,x;
```

```
  WIRE
     select = VCC, // Habilitamos la memoria para ser leida o escrita
     write = VCC, // Vamos a escribir la memoria, no a leerla
     in[32],
     out[32],
     addr[6];

  for(x=0;x<10;x++) {
    WIRE no_write=GND;
    printf("%d\n",x);
    addr[0]=(x&1)*255;
    addr[1]=((x&2)>>1)*255;
    addr[2]=((x&4)>>2)*255;
    addr[3]=((x&8)>>3)*255;
    addr[4]=((x&16)>>4)*255;
    addr[5]=((x&32)>>5)*255;

    for(k=31;k>=0;k--) if(cadena[x][31-k]=='0') in[k]=GND; else
in[k]=VCC;

    for(i=0;i<20;i++) memoria.run(select,write,addr,in,out);
    escribir_memoria();
    //getchar();
  }//fin del bucle que carga el programa en memoria
  //escribir_memoria();
}
/*************LECTURA DEL ARCHIVO EN CàDIGO MµQUINA*************/
void leer_arch (char *nomfi)
{
    int i;

    FILE *fp;

    if((fp=fopen(nomfi,"rb"))==NULL)
    {
      printf("Error de lectura de archivo\n");
      exit(0);
    }
printf("CONTENIDO DEL ARCHIVO: \n");
  i=0;
  for(;;)
    {
      cout << "variable i: " << i << endl;
      fgets(cadena[i],40,fp);
      if(i==64) break;
      printf("%d %c %s",i,cadena[i][0],cadena[i]);
      i++;
    }
    fclose(fp);

    printf("\n");
}

/***************FIN DE LECTURA****************/

/*******************EJECUCION DEL MULTICICLO******************/
void ejecutar(int iter,int longitud)
{
  int i,j;
  MemWrite = GND;
```

```
  WIRE uno[32] = {255,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0,
             0,0,0,0,0,0,0,0};



/***************************************************************/
  /************************************************************/

printf("CONTENIDO DE MEMORIA ANTES DE EJECUTAR.\n");
  escribir_memoria();
  printf("duraci'on del semiciclo de reloj: %d\n",longitud);
  //bucle de simulacion
  for(i=0;i<longitud;i++) reloj=VCC;
  for(j=0;j<iter;j++)
  {
    cout << "iteracion=" << j << endl;
    /******reloj del sistema*********/
     Clock(j,longitud,reloj);
     cout << "reloj=" << (int)reloj << '\n';
    /******unidades funcionales**********/
     printf("MemWrite en programa= %d\n",MemWrite);
     memoria.run(seleccion,MemWrite,out_IorD,dato2,out_memoria);
     cout<<"Salida de memoria:";
     for(i=31;i>=0;i--)cout<<"   "<<(int)out_memoria[i];
     cout<<"\n";

     procesador.run(reloj,out_memoria,MemWrite,out_IorD,dato2);

  }//fin del bucle de simulacion

printf("\nCONTENIDO DE MEMORIA DESPUES DE EJECUTAR.\n");
escribir_memoria();

}//fin de la ejecucion

void escribir_memoria()
{
  int i,k;
  WIRE
     select = VCC, // Habilitamos la memoria para ser leida
     write = GND, // Vamos a leer la memoria, no a escribirla
     in[32],
     out[32],
     addr[6];

  for(int x=0;x<10;x++)
  {//escribe en binario direcciones de 0 a 64 que representan
direcciones de
      //la memoria
    addr[0]=(x&1)*255;
    addr[1]=((x&2)>>1)*255;
    addr[2]=((x&4)>>2)*255;
    addr[3]=((x&8)>>3)*255;
    addr[4]=((x&16)>>4)*255;
    addr[5]=((x&32)>>5)*255;
   //va copiando en la memoria todas las variables
    for(int i=0;i<40;i++) {
      memoria.run(select,write,addr,in,out);
    }
```

```
    //coge los ceros y unos de la cadena y los convierte en variables
    //utiles a la memoria(gnd o vcc)
    cout << setw(2) << x << ' ';
    for(k=31;k>=0;k--) {
      if(out[k]==GND) cout << '0';
      else cout << '1';
    }
    cout << '\n';

  }//fin del bucle que lee el programa de memoria
 printf("\n");

}
```

- **Programa de prueba.**

  - **en binario.**

```
10001100000010100000000000000101
10001100000010110000000000000101
00000001011010100110000000100000
10101100000011000000000000000110
00001000000000000000000000000010
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000001
00000000000000000000000000000001
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

- **en ensamblador:**

```
lw    $10,5($0)
lw    $11,5($0)
add  $12,$11,$10
sw    $12,6($0)
```

Si al final de la simulación aparecía un 2 en la posición 6 de memoria, la simulación era correcta. Cosa que sucedió.

• Dificultades en la implementación.

Han sido muchas las dificultades que se nos han presentado a la hora de realizar la implementación del simulador, desde dificultades meramente técnicas a dificultades de programación. Pero la principal dificultad surge por la necesidad de cambiar de forma de pensar.

Nosotros siempre hemos estado acostumbrados a pensar de forma teórica, según los caminos de datos estudiados en arquitectura de computadores. Pero esto no se ajusta a la realidad.

En la realidad, nosotros tenemos que concebir el simulador como si de componentes reales se tratase. Es decir, como si tuviésemos una "pastilla" para el procesador, otra para la memoria, y los buses para contectarlos entre sí.

Solo podremos trabajar con los componentes y buses públicos, siendo los componentes internos inaccesibles, como ocurre en la realidad.

Esto es algo esclusivo de la programación orientada a objetos, por eso nos resulta tan útil para nuestro propósito.

Seguidamente, puntualizamos todas las dificultades que hemos tenido:

- Tuvimos que crear diversas unidades que no fueron suministradas por el profesor; una unidad de control del multiciclo, una unidad de control de la alu, una unidad de signo extendido, una unidad de desplazamiento a la izquierda y una unidad de desplazamiento a la izquierda que suma los 4 bits mas significativos del PC.

- Crear los buses, cada uno de su tamaño específico.

- Los multiplexores son de 32 bits, y para facilitar la asignación de buses a estos, modificamos la llamada al método de forma que ubiese una entrada por cada bus de entrada, como ocurre en la realidad.

- Los registros necesitan de la señal de reloj para funcionar.

- La habilitación de escritura de los registros, es activa por flanco, por eso nos hemos visto obligados a conectar una puerta *and* a la línea de control y al reloj para habilitar cada uno de los registros, y también al fichero de registros.

- Forzamos al reloj a que empieze en GND haciéndolo correr durante la longitud del ciclo a VCC, pero esta modificación al final no tuvo consecuencias.

- Para enviar la señal de escritura a la memoria fue necesario hacer una asignación a la variable pública de habilitación.

- Tuvimos muchas dificultades para depurar el control, por el gran nº de puertas y conexiones que contiene y por lo dicho anteriormente de entender el concepto de variables públicas y variables privadas.

- Conectar unas unidades con otras nos llevó mucho tiempo por la multitud de conexiones que se deben realizar. Para verlo solo basta con mirar el dibujo.

- En las conexiones entre unidades, no se deben utilizar reasignaciones a otros cables (puenteos), ya que al no ser el mismo cable no sabemos ni a donde va ni de donde viene, ni si sus valores se verán afectados en el transcurso de la simulación.

- Cuando comenzábamos la simulación, los registros comenzaban en unos valores indeterminados que producían datos erróneos, y entonces nos vimos forzados a inicializarlos a 0 utilizando constructores de clase.

- En la segunda etapa del multiciclo, la linea de control 3, no permanecía activa como debiera, suministrando entonces un valor erróneo a la alu. Investigamos entonces en la unidad de control, y llegamos a encontrar un fallo en la implementación de esta, que habíamos tomado del libro ORGANIZACIÓN Y DISEÑO DE COMPUTADORES de Hennessy-Patterson. Lo corregimos incorporando una nueva entrada procedente de la puerta and 1 a la puerta or correspondiente con la linea de control 3.

- Tambien nos fallaba la alu, pero era porque las líneas de salida del control de la alu no estaban bién conectadas a las correspondientes entradas de la alu.

- Hacer el programa de prueba, en binario, tambien nos costó un poco, todo hay que decirlo.

4. Posibilidades de aplicación.

Nuestro simulador nos permite observar los estados en todo momento de todos sus componentes así como el estado final de la memoria.

Esto sería muy útil para detectar posibles fallos en el sistema, ejecutando programas de prueba.

Nos permitiría calcular el tiempo de retardo en función del retardo de sus componentes básicos, así como calcular la frecuencia de reloj apropiada para cada tipo de instrucción.

Pero la verdadera utilidad y lo que a nosotos nos sirve es, como ya hemos dicho anteriormente, a comprender el funcionamiento real de un microprocesador, pudiendo ver paso a paso el estado de sus componenetes al ejecutarse un programa.

Aunque la construcción de nuestro simulador ha sido muy laboriosa, tan solo está preparado para ejecutar las instrucciones básicas de MIPS, como son la lw, sw, add, beq y j.

Otro posible trabajo a realizar sería la incorporación del resto de instrucciones del repertorio MIPS, lo que requeriría múltiples ampliaciones y cambios en el microprocesador, pero esto será otra historia.

- **Autores:**

Diego Jesús Martínez Salvador.
Salvador Herrada Camacho. ***I.T.I.S.***

Agradecemos la colaboración del profesor Vicente Gonzalez Ruiz, sin el cuál no hubiese sido posible la realización de este simulador.