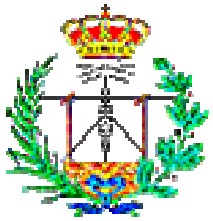


UNIVERSIDAD DE JAÉN

ESCUELA UNIVERSITARIA POLITÉCNICA DE LINARES

Departamento de Electrónica

Área de Ingeniería Telemática

CURSO ACADEMICO 2001/02

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE HERRAMIENTAS DE DESARROLLO PARA SÍMPLEZ Y ALGORÍTMEZ

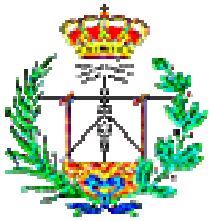
Titulación: **Ingeniería Técnica de Telecomunicación**

Especialidad: **Telemática**

Alumno: **José Luis Blanco Claraco**

Tutor: **Sebastián García Galán**

Linares, Julio de 2002



UNIVERSIDAD DE JAÉN

ESCUELA UNIVERSITARIA POLITÉCNICA DE LINARES

Departamento de Electrónica

Área de Ingeniería Telemática

Diseño e implementación de herramientas de desarrollo para Símplez y Algorítmez

REALIZADO POR:

José Luis Blanco Claraco

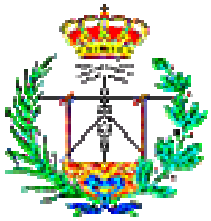
DIRIGIDO POR:

Sebastián García Galán

PALABRAS CLAVE: Simulador. Java. Símplez. Algorítmez. Entrenador.

RESUMEN: Se han diseñado e implementado herramientas software que permiten el ensamblado y simulación de programas de Símplez y Algorítmez, así como un entrenador hardware que emula al microprocesador Algorítmez y sus periféricos.

Linares, Julio de 2002



UNIVERSIDAD DE JAÉN

ESCUELA UNIVERSITARIA POLITÉCNICA DE LINARES

Reunido el Tribunal Calificador en el día de la fecha, constituido por:

D._____

D._____

D._____

Para juzgar el Proyecto Fin de Carrera titulado:

Diseño e implementación de herramientas de desarrollo para Símplez y Algorítmez

del alumno D. José Luis Blanco Claraco
dirigido por D. Sebastián García Galán

HA ACORDADO POR _____ OTORGAR LA CALIFICACION

DE _____

y, para que conste, se extiende firmada por los componentes del Tribunal la presente diligencia.

Linares, a ____ de _____ de _____

El secretario	**El presidente**	**El vocal**
Fdo._____	Fdo._____	Fdo._____

ÍNDICE DE LA MEMORIA

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

Debido a la creciente complejidad de los microprocesadores, microcontroladores y en general de los sistemas digitales, cada vez se hace más necesario el uso de simuladores software de dichos sistemas, con el fin de ayudar en la tarea del desarrollador para entender el funcionamiento de dichos sistemas, verificar y depurar sus diseños. Además, una vez que se quiere probar un diseño en el microprocesador real, es muy útil el uso de los llamados entrenadores, sistemas especialmente diseñados para la depuración de aplicaciones, pero a diferencia de los simuladores, sobre un sistema hardware real.

Siguiendo estas ideas, se van a desarrollar herramientas de estos dos tipos para los microprocesadores Algorítmez y Símplez que, si bien no existen comercialmente, son una buena introducción para microcontroladores más complejos, principalmente con fines didácticos.

2. Objetivos

Con este proyecto se pretende diseñar e implementar unas herramientas software y hardware útiles para la docencia e impartición de prácticas de la asignatura Fundamentos de Computadores I de la E.U.P. Linares. Para ello se crearán tres sistemas software y hardware:

- Simulador software de Símplez.
- Simulador software de Algorítmez.
- Entrenador hardware de Algorítmez. Al no existir comercialmente dicho procesador, se emulará sobre un microcontrolador de propósito general.

Además, se pretende que el alumnado pueda acceder fácilmente a las herramientas software desde una web, e incluso ejecutar los simuladores desde ésta.

3. Herramientas utilizadas

Debido a la distinta naturaleza de cada aplicación, se enumeran por separado las herramientas necesarias para desarrollar cada una de ellas:

3.1. Simuladores

Para los simuladores se ha elegido usar la tecnología Java, debido a las siguientes ventajas:

- Portabilidad. Las aplicaciones se podrán ejecutar tanto en Windows, Linux o cualquier otra plataforma que soporte un mínimo de requisitos (Ver apéndice A1: “Requisitos del sistema”)
- Fácil distribución. Un aspecto importante a tener en cuenta debido a su orientación al uso en prácticas, es lo fácil que resulte distribuir estas aplicaciones en un gran número de máquinas. Si una máquina tiene acceso a una red TCP/IP se pueden colocar los simuladores en un servidor web, con lo que se soluciona el problema de la instalación de la aplicación en muchas máquinas.
- Interfaz gráfica independiente de la plataforma.

Las herramientas que se han usado son principalmente:

- Entorno JBuilder 6.0 de Borland, para la creación de los applets, cuadros de diálogo y edición del código Java, así como su compilación y depuración.
- Microsoft SDK for Java 4.0. Para la creación del certificado de seguridad y el firmado con este de los *applets* para el navegador Internet Explorer. Además, como este SDK incluye un compilador de Java a formato EXE para la plataforma Windows, se ha compilado una versión de los simuladores idéntica a la accesible con los navegadores, pero para ejecución offline.

3.2. Entrenador

El entrenador consta de dos partes:

- El entrenador en sí. Consiste en un sistema hardware basado en microcontrolador capaz de emular de forma autónoma el funcionamiento del microprocesador Algorítmez y sus periféricos. Para el desarrollo del firmware del microcontrolador (ensamblado y depuración del mismo), se ha usado la herramienta AVRStudio 3.5 de la compañía Atmel, disponible gratuitamente en la web del fabricante: http://www.atmel.com

- El comunicador. Es una aplicación que se ejecuta en un PC y se comunica con el entrenador a través de un puerto serie. Debido a las necesidades de ésta aplicación de acceso a los recursos hardware, se ha optado escribirlo en lenguaje C++, usando el entorno Borland C++ Builder 5 para Windows, usando para el ensamblado de programas fuente Algorítmez otro programa adicional en Java, como se describirá en la memoria.

4. Descripción del proyecto

En las siguientes secciones se tratan cada uno de los desarrollos de que consta este proyecto. Primero se analizarán los dos sistemas software simuladores de Símplez y de Algorítmez para pasar después al sistema entrenador y todas las aplicaciones y diseños que han sido necesarias desarrollar para su funcionamiento.

4.1. Simulador de Símplez

Se ha desarrollado un software que es capaz de ensamblar programas escritos para el procesar Símplez, así como de simular la ejecución de estos, siguiendo las especificaciones de [1- Gregorio Fernández].

Una de las características más interesantes de este emulador es que se simula el funcionamiento de la micromáquina (frente a la mas sencilla emulación de la máquina convencional), permitiendo incluso reprogramar el contenido de la memoria de control, modificación de las instrucciones, etc...

4.1.1. Especificación de la máquina Símplez

4.1.1.1. Arquitectura y modelo de programación

Características de este procesador:

- Arquitectura básica Von Neumann.
- Sólo 8 instrucciones.
- Longitud de palabra de 12 bits.
- Espacio de memoria y E/S direccionable de 9 bits (512 palabras máximo)
- No soporta interrupciones.
- Dos periféricos: Un teclado y una pantalla.

El modelo elegido para implementar la máquina es el modelo microprogramado, en el cuál las señales de control de la UCP se generan a partir de una MC (memoria de control) interna al secuenciador, como se ve en la siguiente figura.

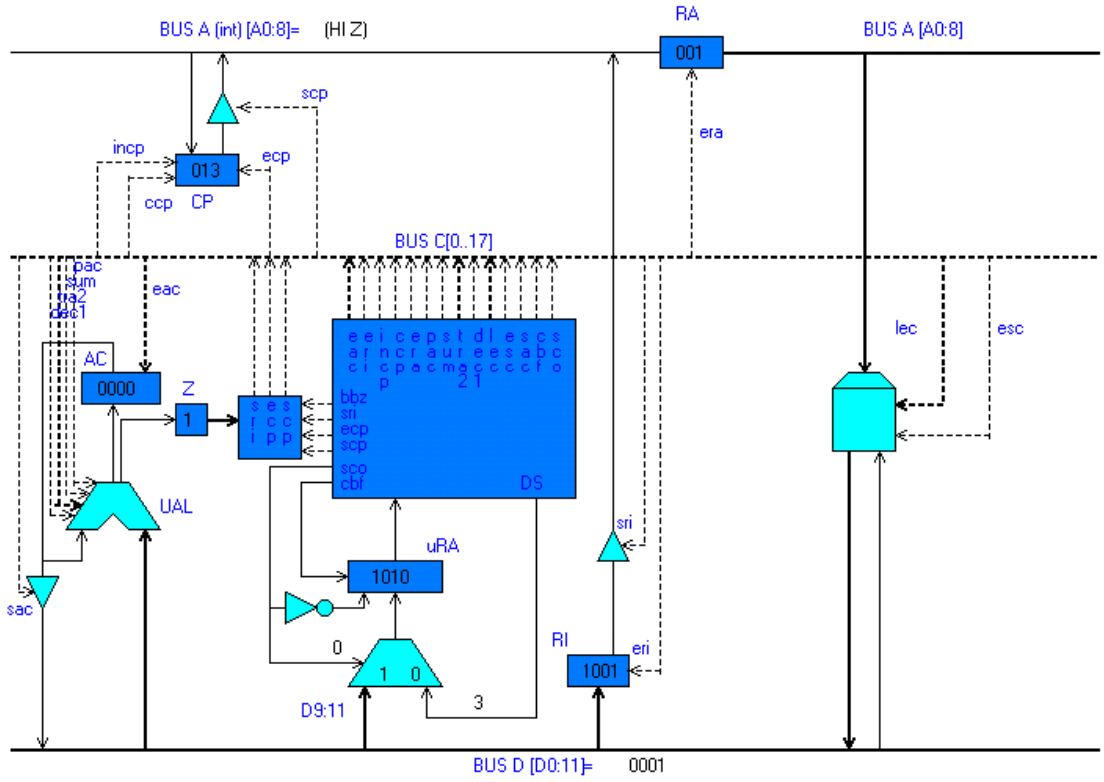


FIGURA 1. Elementos de micro-Símplez

Los elementos que componen la máquina, detalladamente, son:

- UCP (Unidad central de proceso). Es el procesador Símplez en sí. Su interfaz al exterior son el bus A (bus de direcciones), el bus D (bus de datos) y dos líneas del bus C (bus de señales de control), las señales LEC y ESC, usadas para entradas y salidas con los dispositivos externos. La UCP la componen:
    - La UAL. Es el elemento que realiza las operaciones aritmético lógicas. En el caso de un procesador tan sencillo como Símplez, solo realiza 4 operaciones: puesta a cero, decremento, suma y carga desde el bus D.
    - Registros. Solo existe uno, el acumulador, que recibe los resultados de la UAL.
    - MC (Micro-memoria de control) y registro uRA (registro de dirección de micro-memoria de control), que ejecutan el microprograma almacenado para cada instrucción del modelo de máquina convencional.
- Elementos externos. Se representan como una “caja negra” con acceso a los buses A, D, y las señales LEC y ESC. Dependiendo de la dirección en el bus D, se accede a dispositivos diferentes:
    - 0-507 (Q’000-Q’773). Es memoria RAM. Aquí es donde se almacena el programa en ejecución, así como todas las variables y constantes necesarias.
    - 508-509 (Q’774-Q’775). Controlador de pantalla.
    - 510-511 (Q’776-Q’777). Controlador de teclado.

4.1.1.2. Formato de instrucciones

Como se ha visto arriba, el tamaño de palabra de Símplez es de 12 bits. Además, todas las instrucciones son de una sola palabra, frente a otros modelos como en Algorítmez donde la longitud es variable. Se usan solo dos campos en las instrucciones:

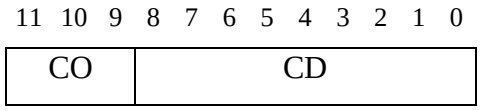


FIGURA 2. Formato de instrucciones en Símplez

Donde:

- CO (Código de operando). Es un código de 3 bits que indica la instrucción a ejecutar. En la sección 4.1.1.3 se ven las instrucciones por defecto.

- CD (Campo de dirección). Indica la dirección del operando sobre el que opera la instrucción. Como se ve, es del mismo tamaño que el bus de direcciones (9 bits), por lo que todas las instrucciones pueden acceder a todas las palabras de memoria de forma directa, no existiendo otro modo de direccionamiento que éste.

4.1.1.3. Instrucciones y modificación del microprograma.

Como se explica en la lección 11 de [1- Gregorio Fernández], la ejecución de cada instrucción requiere 2 o 4 ciclos de reloj, teniendo cada ciclo asignada la activación de una serie de microseñales de forma que el resultado conjunto sea el esperado de la instrucción. La forma de codificar este microprograma es en forma de una memoria de control, de 16 palabras de 21 bits. El contenido por defecto, o estándar, de esta micro-memoria usado por el simulador es el descrito en la citada referencia. Este se puede ver y modificar desde el editor de código ensamblador del programa, pulsando el botón "*Microprogramación*", con lo que se abrirá el editor de modelos de microprogramación:

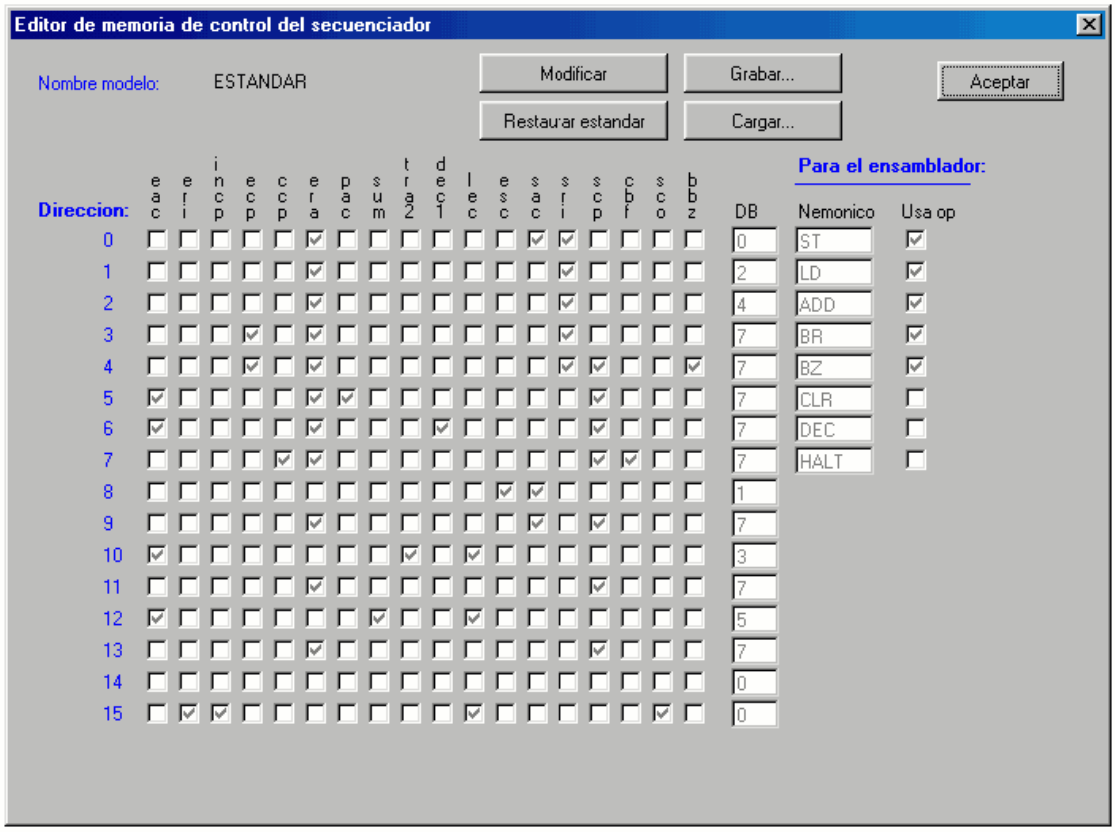


FIGURA 3. Editor de microprogramación.

Se permite la modificación de esta memoria, por ejemplo, para definir nuevas instrucciones. Para facilitar esto, se puede modificar el mnemónico asociado a cada CO en la columna de la derecha, además de indicar que instrucciones necesitan o no un operando. Aunque esta información no esta incluida en la memoria de control es útil para el programador de Símplez y para el programa ensamblador.

El contenido que se ve en la figura 3 es la versión estándar, y contiene las siguientes instrucciones:

- ST (CO=0, Store). Guarda el contenido del acumulador en la dirección indicada en el operando.
- LD (CO=1, Load). Carga en el acumulador el contenido de la dirección indicada en el operando.
- ADD (CO=2, Add). Suma al acumulador, el contenido de la dirección indicada por el operando.
- BR (CO=3, Branch). Salta a la dirección indicada por el operando, incondicionalmente.
- BZ (CO=4, Branch if Zero). Salta a la dirección indicada por el operando, solo si el indicador Z esta activado. Sino, sigue ejecutando normalmente.
- CLR (CO=5, Clear). Carga un cero en el acumulador.
- DEC (CO=6, Decrement). Decrementa en una unidad el acumulador.
- HALT (CO=7, Halt). Paraliza el estado de la máquina.

4.1.2. Ensamblador

4.1.2.1. Lenguaje ensamblador

Las normas de los programas en ensamblador para Símplez son:

- Cada línea puede contener una sola directiva, instrucción o pseudo-instrucción.
- No es obligatorio el uso de indentaciones.
- Puede usarse una etiqueta opcional en cada línea.
- Entre cada expresión debe haber al menos un espacio.
- No se distinguen mayúsculas de minúsculas.
- Las etiquetas deben comenzar por una letra.

Las directivas permitidas en Símplez y sus formatos son:

- [etiqueta] ORG <dirección 9 bits>

  Lo que indica al ensamblador que continué almacenando el código producido en la dirección indicada. No es obligatorio su uso. Por defecto, se comienza almacenando el código en la dirección Q'000, el vector de reset.

- [etiqueta] END

  Indica al ensamblador que ha llegado al final del código fuente. Su uso es obligatorio.

Las pseudo-instrucciones permitidas son:

- [etiqueta] DATA <constante 12bits> [, <constante 12bits> …]

Guarda directamente los valores indicados en la posición de memoria actual. La etiqueta referenciará siempre la dirección del primer elemento. Se pueden usar cuantas

expresiones separadas por el signo coma (“,”) como se quieran, incluso cada uno en un formato distinto. Un ejemplo:

```
        BR      /INI
DATO    DATA    INI,7,"A",H'7F,B'01101001
INI     LD      /DATO
```

La sentencia DATA del ejemplo, genera las siguientes palabras en memoria, a partir de la dirección 1:

[1]= 6 (Valor de la etiqueta INI)
[2]= 7 (Valor decimal)
[3]= 65 (Valor ASCII del carácter)
[4]= 127 (Valor en hexadecimal H’7F)
[5]= 105 (Valor en binario B’01101001)

- [etiqueta] RES <expresión numérica>

  Indica que se reserven el numero de palabras indicadas en la memoria. La etiqueta referenciará siempre la dirección de la primera palabra.

Por último, las instrucciones se introducen siguiendo la sintaxis:

- [etiqueta] <NEMONICO> [/<dirección 12 bits>]

Donde el uso o no de campo de dirección depende de cada instrucción. En Símplez solo existe direccionamiento directo.

4.1.2.2. Expresiones numéricas

En los ficheros fuente, cualquier valor numérico se considerará por defecto en decimal, pero se pueden usar los siguientes prefijos para cambiar la base:

Prefijo	Base
B’	Binario
Q’	Octal
D’	Decimal
H’	Hexadecimal

TABLA 1. Prefijos numéricos en Símplez

4.1.2.3. Implementación del ensamblador

Tanto el ensamblador como el emulador de Símplez se han encapsulado en una misma clase Java, cuyos elementos principales se ven en el siguiente esquema:

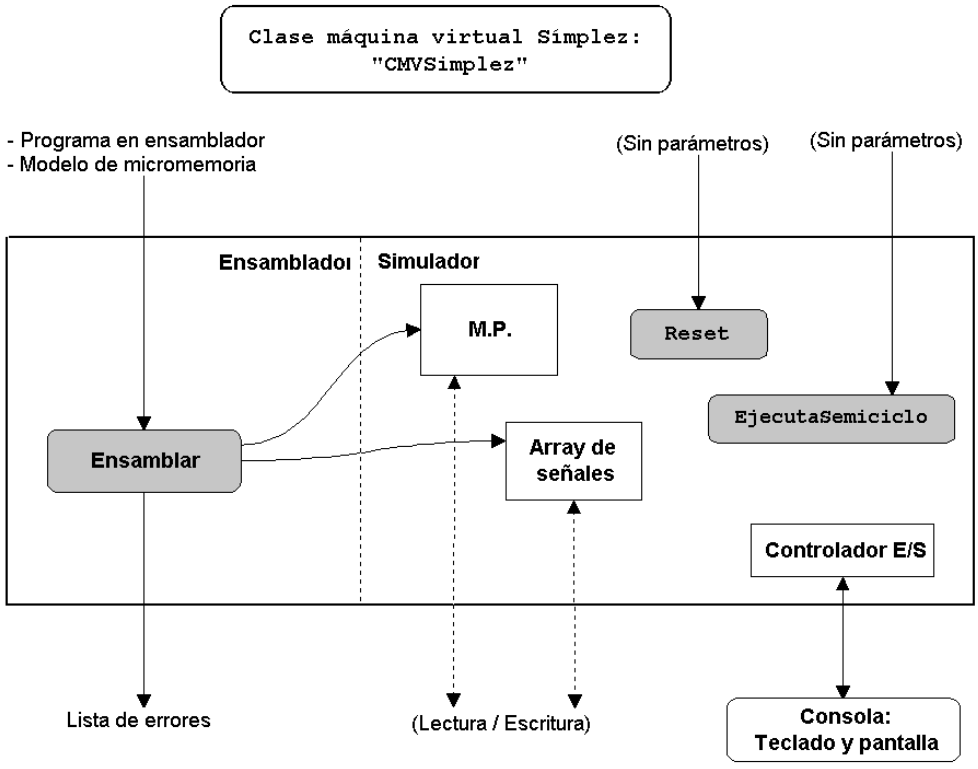


FIGURA 4. Esquema de la clase *CMVSimplez*

Donde ven que los tres métodos públicos principales que esta clase ofrece son:

- Ensamblar.
- Reset.
- EjecutaSemiciclo.

En este punto solo se va a tratar el primer método, el ensamblador de programas. Llamando a este método se transforma el programa escrito en código ensamblador Símplez pasado como argumento, en código máquina ejecutable por el simulador.

Además, se pasa un modelo de micro-programación, almacenado en una clase Java de tipo *CMicroModelo*, donde se almacenan los siguientes datos sobre el modelo definido por el usuario:

- Mnemónicos: Nombre de los 8 mnemónicos correspondientes a los posibles valores del CO de una instrucción.
- Memoria de control: Un vector de 16 palabras de 21 bits. Definen el comportamiento de cada instrucción.
- Indicador de operando: Uno para cada instrucción, indicando si usa o no el campo de dirección.

Como se muestra en la figura 3, todos estos datos son accesibles desde un diálogo en el editor de programas.

Uno de los principales problemas para un ensamblador, son las referencias a etiquetas aún no declaradas. Para solucionarlo, se realiza un ensamblado en dos pasadas, de forma que en la segunda pasada ya conocemos todas las etiquetas y sus valores. En el caso de Símplez el problema se simplifica algo al ser todas las instrucciones de la misma longitud. Así, en una primera pasada solo hay que analizar cada línea si se trata o no de una instrucción, pseudo-instrucción o directiva, e ir creando la lista de etiquetas.

El algoritmo usado por el ensamblador, descrito en lenguaje natural, es el siguiente:

1. Para pasada = 1..2 ejecutar:
    1.1. Para cada línea del programa:
        1.1.1. Preprocesar línea: Quitar tabuladores, comentarios y dejar solo un espacio entre cada palabra.
        1.1.2. Separación de *tokens*. Se extraen de la línea hasta 3 palabras separadas por espacios.
        1.1.3. Intentar interpretar primer *token* como pseudoinstrucción, instrucción o directiva. Sino coincide, probar con el segundo *token*. Si éste si coincide, se añade una nueva etiqueta con el nombre del primer *token* y con la dirección actual.
        1.1.4. Al interpretar una instrucción, usar los mnemónicos proporcionados por el *CMicroModelo*. Si se usa en el campo de dirección una etiqueta desconocida:
            1.1.4.1.Primera pasada: Ignorar línea y pasar a la siguiente.
            1.1.4.2.Segunda pasada: Lanzar error de etiqueta desconocida.

Para los métodos que se encargan del ensamblado de cada línea se usa un sistema de excepciones Java, de forma que cualquier error en la sintaxis se pueda devolver fácilmente en forma de excepción por el método principal de ensamblado, lo que simplifica y clarifica el programa al evitar continuas comprobaciones de situaciones de error: Se supone siempre que no ocurren errores, ya que cuando alguno se encuentra se lanza una excepción y no se sigue intentando ensamblar esa línea.
Además, durante la segunda pasada se guarda la relación línea de código-posición de memoria, de forma que sea posible en el simulador ver que línea de código fuente corresponde a cada posición de memoria.

El resultado del ensamblado se guarda directamente en la memoria de Símplez, en lugar de en un fichero objeto. Esto es así porque puede que el usuario no quiera conceder los privilegios de acceso a ficheros al applet. Así se asegura un mínimo de funcionalidad incluso con las mayores restricciones.

4.1.3. Simulador

4.1.3.1. Funcionamiento

El simulador se ha creado teniendo en cuenta todos los detalles de la micromáquina de Símplez, de forma que realmente se simulan todas las líneas de control y registros. En cada momento, el estado de la máquina queda perfectamente definido por el contenido de la memoria RAM y un conjunto de señales, registros y buses, como se muestra en las figuras 1 y 4. El array de señales, contiene todos los valores de registros, señales y buses en cada instante, incluyendo el reloj de entrada al microprocesador. La lista completa se resume a continuación:

Nombre	Descripción / Acción	Número de bits	Señal	Registro	Bus
eac	Entrada en acumulador	1	X		
eri	Entrada en registro de instrucción	1	X		
incp	Incrementa contador de programa	1	X		
ecp	Entrada en contador de programa	1	X		
ccp	Pone a cero el contador de programa	1	X		
era	Entrada en el registro de dirección (RA)	1	X		
pac	Pone a cero el AC	1	X		
sum	Suma al AC el valor en el BUS D	1	X		
tra2	Pone en el AC el valor en el BUS D	1	X		
dec1	Decrementa en una unidad el valor del AC	1	X		
lec	Lectura de dispositivo externo (memoria o E/S)	1	X		
esc	Escritura en dispositivo externo (memoria o E/S)	1	X		
sac	Salida del AC al bus D	1	X		
sri	Salida del registro de instrucción al BUS A interno	1	X		
scp	Salida del contador de programa	1	X		
cbf	Permite modificación de uRA si vale 0.	1	X		
sco	Indica que el valor del CO de la instrucción en RI debe ser los 3 bits bajos de uRA en el siguiente ciclo.	1	X		
bbz	Enmascara señales según valor de Z.	1	X		
R	Señal de reloj.	1	X		
Z	Indicador de cero.	1		X	
uRA	Contador de dirección actual en la micromemoria de control.	4		X	

AC	Acumulador	12		X	
CP	Contador de programa	9		X	
RI	Registro de instrucciones	12		X	
BUSA	Bus de direcciones interno a la CPU	9			X
BUSD	Bus de datos	12			X
RA	Se toma como un registro y ademas como el valor del bus de direcciones externo	9		X	X

TABLA 2. Señales y registros en Símplez

Todos estos valores se han definido como tipos **int** de Java, es decir, números de 32 bits. Ahora bien, algunas de estas señales son de 12 bits, otras de 9 y otras son binarias, así como algunas son buses y pueden estar en alta impedancia (HI-Z). Para implementar esto se ha declarado la siguiente clase que representa a una señal:

```
package simplez;

class CSignal
{
  public static int HIZ=0x80000000;   // Bit que indica que el
bus esta en HIZ

  String      nombre;
  int         nBits;
  private int valor;

  // Constructor
  public CSignal( String Nombre,int nbits,int valorIni  )
  {
    nombre=Nombre;
    nBits=nbits;
    set(valorIni);
  }

  // Coger valor:
  public int get()  {  return valor; }
  public String getStr() { return Integer.toString(valor); }

  // Poner valor
  public void set(int val)
  {
    valor=val;
  }

}
```

El array que se mostraba en la figura 4 es un array de objetos de esta clase. Como se ve, se usa como una simple estructura de datos. Los datos que se guardan de cada señal son:

- **Nombre**: La descripción de la señal en forma de cadena. Programar toda la máquina de Símplez usando números de indices para cada señal habría sido muy engorroso, propiciaría los errores y haría muy difícil el mantenimiento del código. Por ello, las señales se referencian por su nombre. Como se ve más abajo, esto no representará una pérdida de eficiencia significativa.
- **Número de bits**: El número de bits del registro / bus / señal.
- **Valor**: El valor de la señal en sí. Además, si una señal es un bus, se aprovecha que en Símplez nunca se van a usar los 32 bits de un **int** para el valor, usando el bit mas alto (`HIZ=0x80000000`) como máscara que indica que un bus está en alta impedancia.

La forma de leer / escribir en el array de señales, ya sea desde dentro de la clase CVMSimplez o desde fuera, es usando los métodos:

```
int getSignal(String nom)
void setSignal(String nom,int val)
```

Donde las señales se referencian por sus nombres (“CP”, “AC”, “uRA”, etc...). Para hacer esto eficiente, se ha usado un objeto del tipo `HashTable` para relacionar cada nombre de señal con su índice numérico en el array. Este tipo de tablas usa lo que se llama una función hash, que tiene la propiedad de, dado un objeto cualquiera, devolver un número supuestamente en relación biunívoca para cada objeto. De esta forma, la cadena del nombre de la señal, se convierte en un número lo que evita el realizar las costosas operaciones de comparación de cadenas en Java. Todo esto queda oculto al programador, al hacerlo internamente la misma clase `HashTable`.

Volviendo a los tres métodos públicos principales que la clase CMVSimplez exporta:

- Ensamblar.
- Reset.
- EjecutaSemiciclo.

Para la simulación solamente existen los dos últimos métodos:

- Reset: Reinicia el estado de la máquina virtual. Esto incluye:
  - Acumulador (AC) y contador de programa (CP) a cero.
  - Recargar la RAM con la imagen posterior al ensamblado, de la cual se hizo una copia de seguridad al ensamblar. Esto es muy importante en Símplez, que hace uso extensivo de la auto modificación del programa.
  - Todos los registros a cero, y los buses en alta impedancia.
  - Señal de reloj ("R") en estado alto, y uRA a 15. De esta forma, se coloca a la máquina en el estado común I0: Extracción de la instrucción de la memoria, la que se ejecutará en cuanto se comience a dar pulsos de reloj con el siguiente método:
- EjecutaSemiciclo: Simplemente cambia el valor de la señal de reloj ("R") de alto a bajo o viceversa. A partir de ahí, actualiza todos los elementos del micromodelo según el valor de las señales, de forma que las instrucciones se ejecuten automáticamente:
  - Si en un bus no se envían datos, se pone en hi-z.
  - Si en un registro se ha producido su flanco de entrada correspondiente (de subida o de bajada), se actualiza su contenido con el valor que se presente a su entrada.
  - Si están activas las señales LEC o ESC, distinguir entre direccionamiento a memoria o dispositivo y actualizar en consecuencia.
  - Ejecutar cada una de las microseñales activas INCP, CCP, etc... que representen una acción de cálculo o no sólo de transferencia.

Como se ve, en ningún momento se emulan las **instrucciones** de Símplez, sino solamente las **señales** a un nivel físico. Ahora bien, para ejecutar una instrucción: ¿Cómo sabemos que ha terminado de ejecutarla? Algunas instrucciones emplean 2 ciclos, mientras que otras emplean 4 ciclos. Además, como se puede modificar la micromemoria, no sabemos de antemano la duración de una instrucción. Para esto se ha añadido otro método a la máquina virtual:

```
public boolean EjecutaInstruccion()
{
  do {
    // Un ciclo de reloj:
    EjecutaSemiciclo();
    EjecutaSemiciclo();

    // Ver si CBF:
    if (getSignal("CBF")==1)
    {
      HALT=true;
      return true;
    }

  } while ( getSignal("uRA")!=15 );

  // Ver si estamos en un breakpoint:
  if ( Breakpoints.indexOf(new Integer( getSignal("CP") ))!=-1 )
       return true;
  else  return HALT;
}
```

Donde se ve que el final de la ejecución de una instrucción se detecta por la llegada a la microinstrucción de dirección 15: La etapa de extracción de instrucción común a todas. Además se puede ver como esta función devuelve **true** si hay que parar la ejecución, ya sea por un HALT (lo que en realidad se detecta por la señal *cbf*) o por un breakpoint colocado por el usuario.

4.1.3.2. Interfaz de usuario

Tras ensamblar un programa Símplez, si el código no contiene errores, se abre automáticamente la ventana de simulación:

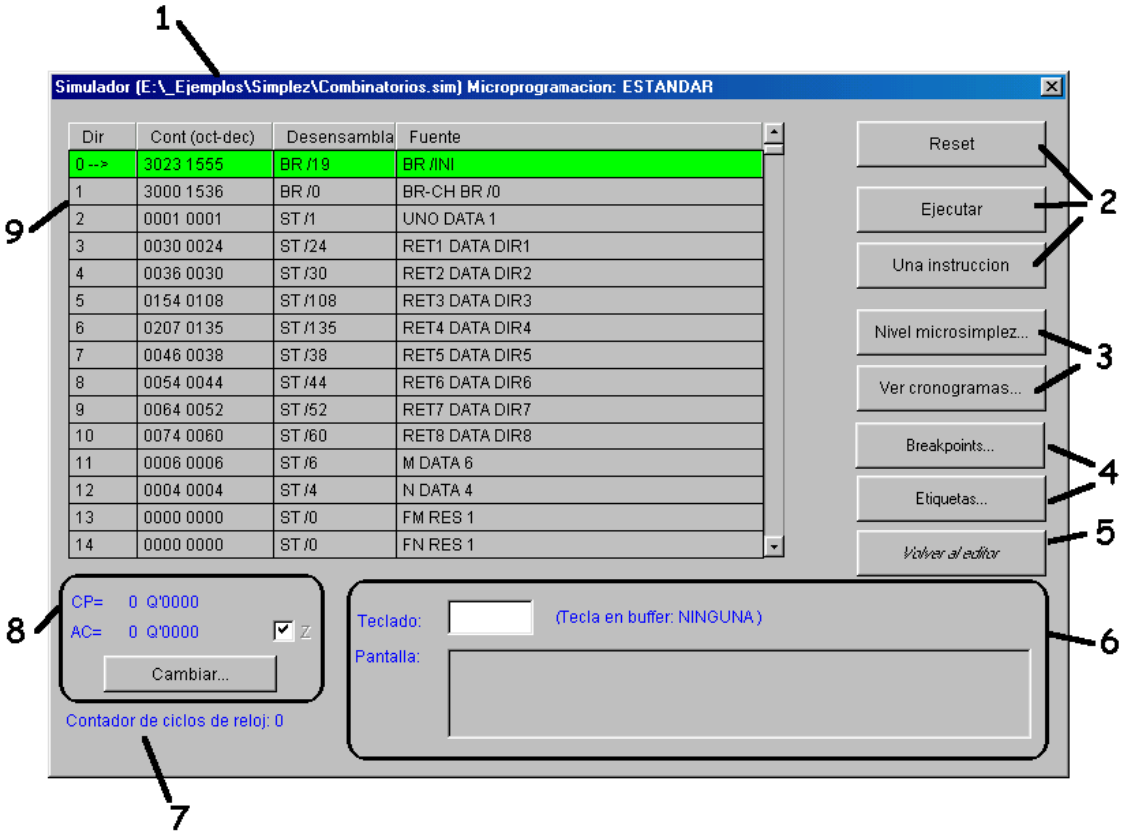


FIGURA 5. Módulo simulador de Símplez

Donde los elementos más importantes son:

1. Título: Donde se puede ver el nombre tanto del fichero fuente ensamblador que se está simulando, como el nombre del micromodelo usado.
2. Botones de control principales:
   - Reset: Llama al método “Reset” de la máquina virtual Símplez, reiniciando el estado y recargando el programa ensamblado en memoria.
   - Una instrucción: Llama al método auxiliar que vimos en el apartado anterior, que se encarga de dar pulsos a la señal de reloj de la máquina virtual hasta que se termine la ejecución de una instrucción completa.
   - Ejecutar: Este botón inicia el modo de ejecución continua: Se ejecutan instrucciones continuamente hasta llegar a un HALT o a un breakpoint. Para ejecutar este modo se crea un hilo de ejecución Java aparte del hilo principal,

implementado en la clase “HiloEjecutor”. Éste se encarga de actualizar periódicamente el estado de la ventana para ir reflejando los cambios de la máquina hasta que se interrumpe su ejecución con el botón “Parar” (Este botón sólo aparece cuando se está en este modo) o alguna de las situaciones mencionadas arriba.

3. Modos de ejecución secundarios. Se ven detalladamente en 4.1.2.3.
4. Botones de:
   - Breakpoints (Puntos de ruptura): Muestra una ventana donde se ven los breakpoints activos, permitiendo quitarlos o añadir nuevos:

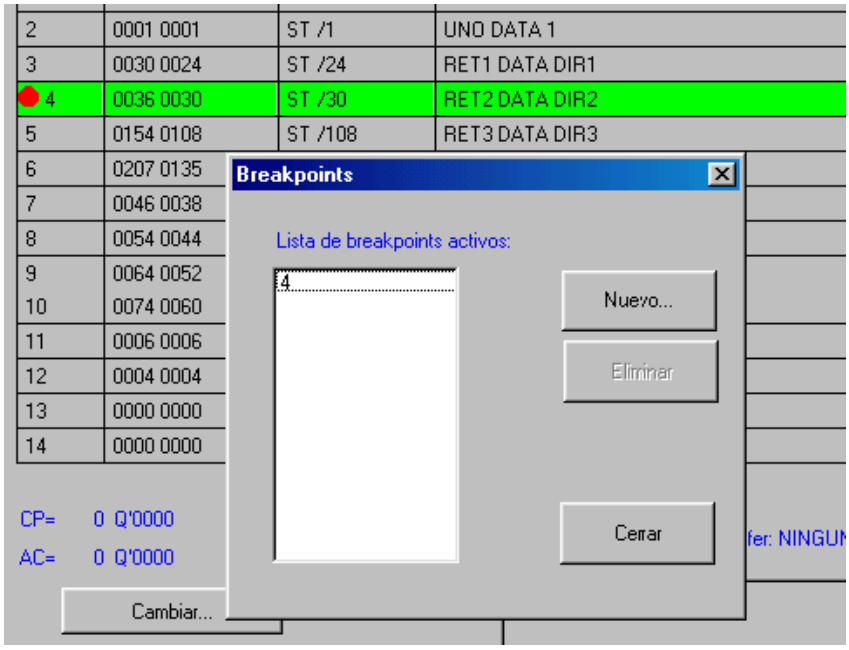


FIGURA 6. Breakpoints en Símplez

   - Etiquetas: Muestra una lista con las etiquetas encontradas en el programa y sus direcciones correspondientes:

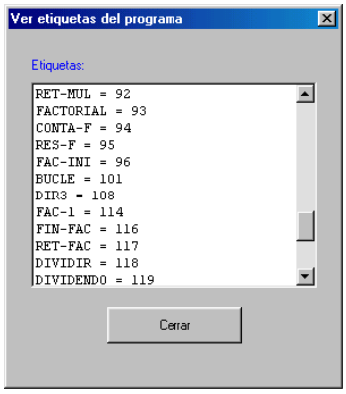


FIGURA 7. Lista de etiquetas

5. Volver al editor: Cierra esta ventana de simulador para volver al editor de programas en ensamblador. Al salir se elimina totalmente el estado de la máquina virtual, incluyendo memoria, puntos de ruptura, etc...
6. Consola de Símplez: Estos controles representan los dos dispositivos de E/S de Símplez: La pantalla y el teclado. Se muestra además el carácter que está preparado por el teclado. Si se pulsa una tecla estando ya el buffer ocupado, la nueva tecla se pierde.
7. Contador de ciclos: Aquí se muestra el número de ciclos de reloj que han transcurrido desde el reset de la máquina virtual. Este contador es llevado internamente por la clase CMVSimplez, por lo que refleja los ciclos ejecutados independientemente de desde qué modulo se han ejecutado (Micromodelo, cronogramas, ventana principal del simulador).
8. Registros: Aquí se puede ver el contenido del acumulador (AC), del contador de programa (CP) y del biestable Z. Además, se permite cambiarlos manualmente en cualquier momento.
9. Tabla de memoria: Aquí se muestran 512 filas, una por cada posición del espacio de direccionamiento de Símplez. Las columnas de izquierda a derecha, representan:
    - Dirección.
    - Contenido de esa palabra de memoria (o de E/S) tanto en decimal como en octal.
    - Desensamblado: Interpretación de esa palabra como una instrucción Símplez, siguiendo el micromodelo seleccionado.
    - Código fuente: La línea del programa original que está relacionada con esa dirección.

4.1.3.3. Modos de ejecución

Dentro del simulador, un programa se puede ejecutar de distintas formas:

- Desde la misma ventana del simulador: Usando los botones “Un paso” y “Ejecutar”.
- Desde el módulo de micromodelo: Pulsando el botón “Nivel microsímplez...” se abre una ventana con una representación gráfica del modelo interno de la CPU Símplez:

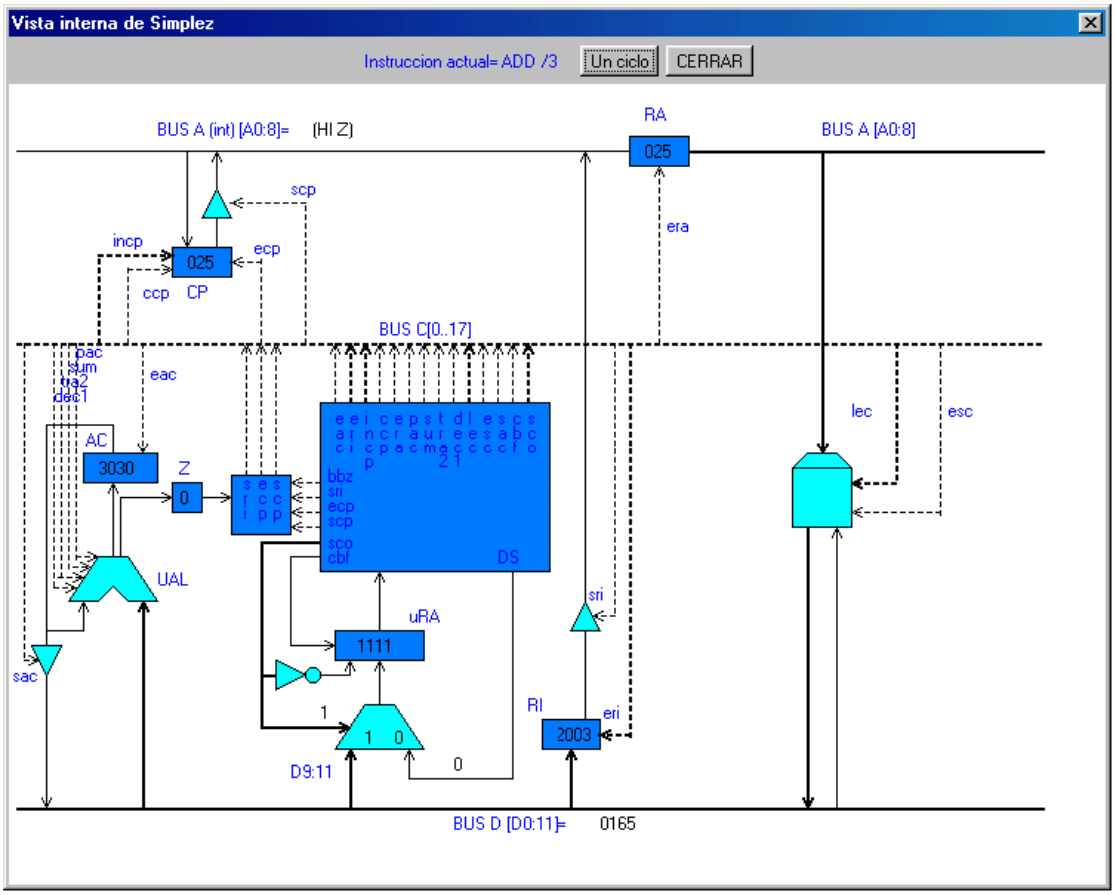


FIGURA 8. Micromáquina de Símplez

Donde se muestra el estado de todas las señales que se listaban en la tabla 2. Pulsando en el botón “Un ciclo” se ejecutará un ciclo de reloj, lo que se realiza llamando dos veces el método “EjecutaSemiciclo” de la clase CMVSimplez.

Para conseguir dibujar todos estos elementos de forma dinámica (si una señal está activada se resalta, se ven los valores de los registros, etc...) se ha diseñado un pequeño renderizador de objetos del tipo “CObjetoGrafico”, usando un formato de

cadenas de texto que describe multitud de objetos gráficos (Cuadros, líneas, textos fijos y variables, etc...). Todos estos objetos de la clase CObjetoGrafico se guardan en un Vector, que será el renderizado finalmente en pantalla. Por ejemplo, la parte del bus de direcciones interno a la CPU se codifica así:

```
// BUS A interno:
objsgraf.addElement(  new CObjetoGrafico(
sim.getSignalIndex("BUSA")  /* Señal */  ,
              "2," +    // Nº de trazos:
              // Cada trazo una linea:
              "5,45, 420,45  ,0,0,"+   // Bus principal
              "120,45,120,110,  0,1"  // Entrada de CP
               )   );

objsgraf.addElement(  new CObjetoGrafico( 200 /* Texto */  ,
              "100,35,'BUS A (int) [A0:8]='") );

objsgraf.addElement(  new CObjetoGrafico( 202 /*Valor en octal*/,
              "205,35,'BUSA',3") );
```

- Desde el módulo de cronogramas: Pulsando en el simulador el botón “Ver cronogramas” se abre esta ventana:

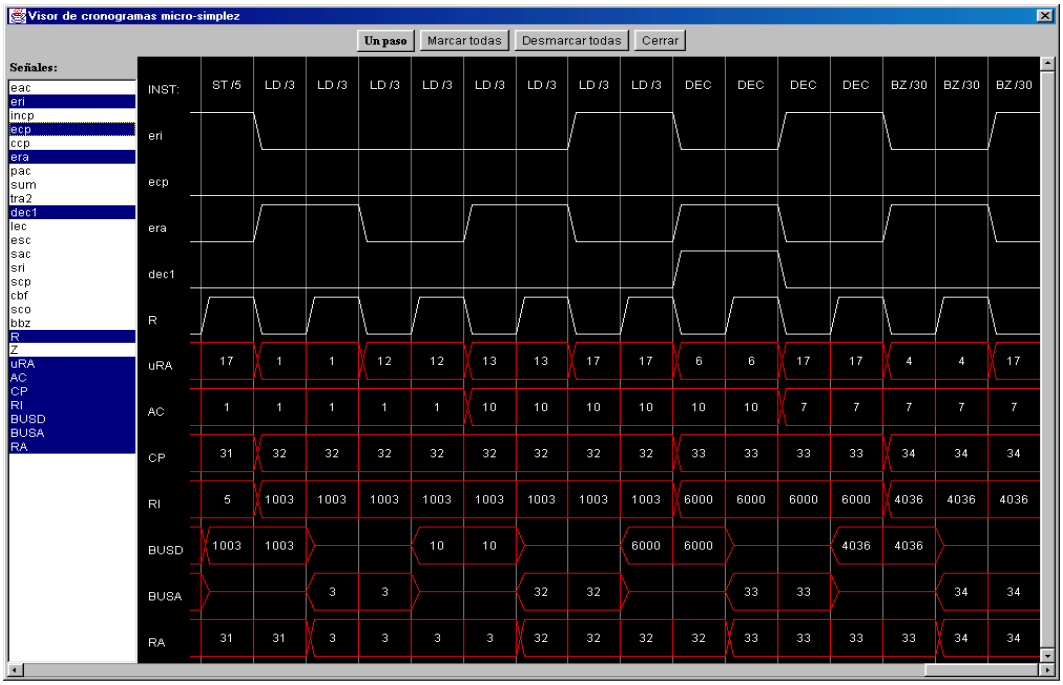


FIGURA 9. Módulo de cronogramas

En este módulo, se puede seleccionar en la lista de la izquierda las señales, buses y registros que se quieren visualizar en cada momento. Pulsando el botón “Un paso”, se ejecutará un ciclo de reloj (dos semiciclos), pudiendo seguir fácilmente todas las transiciones de las señales internas de Símplez. Estas señales son las contenidas en el array de señales del objeto CMSimplez que se listaron en el apartado 4.1.3.1.

4.1.4. Distribuciones

Como se ha especificado en los objetivos del proyecto, los simuladores deben ser fácilmente accesibles, por lo que se han creado en Java siendo accesibles a través de un navegador Web, aunque se han facilitado también en otras versiones para mayor comodidad. Siendo en todos los casos la misma aplicación, se ha preparado de distinta forma dependiendo del medio usado para su distribución.

4.1.4.1. Versión web

De todos los navegadores existentes, muchos incluyen soporte para applets Java: pequeños programas insertados en una página HTML, que se interpretan y ejecutan en el cliente. Pese a esto, existen diferencias significativas entre el soporte que da un navegador y el resto. Recientemente Sun ha desarrollado un sistema que intenta evitar estos problemas, aunque no se ha considerado su uso en este proyecto, ya que requiere la instalación de un plug-in especial dependiente del sistema operativo, además de usar un sistema de certificados de prueba más engorroso de usar para el usuario final.

Los simuladores se han diseñado de forma que funcionen en los dos navegadores de uso más extendido: Netscape Navigator y Microsoft Internet Explorer.
Se encuentran dos problemas principales en cuanto a compatibilidad entre estos dos navegadores para la ejecución de applets:

- Compresión de clases: En todo proyecto que incluya mas de una clase Java, es muy conveniente comprimir todas las clases en un único fichero, de forma que el cliente HTTP pueda descargar con una mayor eficiencia el conjunto de todas las clases, al necesitar una única petición HTTP y además estar comprimidas. Sin embargo, Internet Explorer usa el formato de compresión Cabinet File (.cab) mientras que Netscape Navigator usa el estándar de Sun JAR File (.jar). La solución a este problema pasa por usar ambos sistemas simultáneamente, ignorando cada navegador el fichero y etiquetas que no utiliza.

- Seguridad: Debido a la naturaleza de Internet y a los riesgos que puede conllevar el uso de un applet, una aplicación enviada por un servidor y que se ejecuta en la máquina local, los navegadores imponen por defecto unas normas de seguridad a los applets (aunque estas también varían entre navegadores, pero básicamente son las mismas), ejecutándolos en un llamado *sandbox*, y prohibiendo acceso a ficheros locales, conexiones de red, etc... En este proyecto es necesario el acceso a ficheros para la carga de los programas en ensamblador, por lo que se han usado distintos mecanismos para conseguirlo según el navegador:

  - Para Internet Explorer: Se ha creado un certificado de pruebas, y firmado con él los ficheros CAB, junto con una lista de los derechos que reclama el applet. Para ello se han usado herramientas del "Microsoft SDK for Java 4.0". Primero hay que crear un certificado de pruebas, lo que se realiza con la herramienta *makecert*:

    ```
    makecert -sv miKey.pvk -d "Certificado de Simuladores" –n
    "CN=Jose Luis Blanco" miCert.cer
    ```

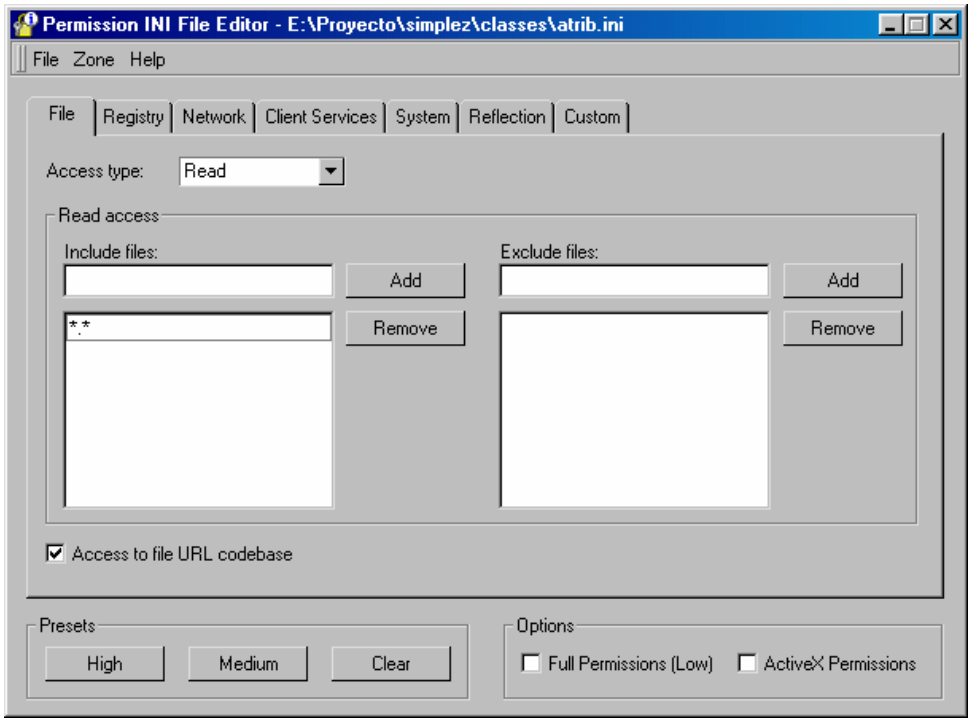
    Lo que creará el fichero *Mikey.pvk* con una clave de cifrado y el fichero *miCert.cer* con el certificado de pruebas. Estos certificados dependen directamente de la "Root Agency" y por lo tanto no tendrían ninguna validez como elementos de seguridad en aplicaciones comerciales. En su lugar, habría que comprar un certificado a entidades como Verisign.
    A continuación se crea un certificado de prueba para publicación (*Software Publisher Certificate*) a partir del certificado creado anteriormente, usando la herramienta *cert2spc* lo que creará el fichero *MiCert.spc* con el certificado listo para firmar:

    ```
    cert2spc MiCert.cer MiCert.spc
    ```

    Antes de realizar la firma, se prepara un fichero "attrib.ini" con los permisos que queremos que el applet reclame. La edición de este fichero se realiza con el programa *piniedit*. En este se puede elegir entre tres

niveles de seguridad predeterminados: Bajo, medio y alto, o crear una configuración personalizada.

En este caso se ha usado una configuración personalizada, solicitando solo los derechos de creación de hilos, ventanas, acceso al portapapeles, y a ficheros.



A continuación se usa la herramienta *cabarc* para comprimir todas las clases del simulador en un fichero CAB:

```
cabarc -p -r N SimSimplez.cab componentes\*.class simplez\*.class
```

La firma en sí la realiza la herramienta *signcode*, que se ha llamado con los siguientes parámetros:

```
Signcode -j javasign.dll -jp "atrib.ini" -spc MiCert.spc -v MiKey.pvk SimSimplez.cab
```

De esta forma el fichero CAB queda firmado digitalmente. Para realizar el *timestamping* del mismo, se ha usado el servicio gratuito que ofrece Verisign a través de su web, con el siguiente comando:

```
signcode -x -t http://timestamp.verisign.com/scripts/
timestamp.dll -tr 5 SimSimplez.cab
```

Lo que incluirá de forma segura, en la firma, la hora y fecha actuales.

- Para Netscape Navigator: Este navegador usa la compresión de clases Java en formato JAR, por lo que se ha usado la herramienta *jar* del JDK1.2 de Sun:

```
jar cfm SimSimplez.jar manifest.mf simplez/*.class
componentes/*.class
```

Además se ha incluido en el fichero JAR un fichero estándar *manifest.mf* indicando la clase principal de la aplicación:

```
Manifest-Version: 1.0
Main-Class: algoritmez.AppletLanzador
Created-By: 1.3.0_02 (Sun Microsystems Inc.)
```

Para solicitar los permisos se han usado la clases de Netscape *PrivilegeManager*, para solicitar en tiempo de ejecución estos al navegador. Antes, el programa se asegura de estar ejecutándose sobre un navegador de Netscape:

```
String tipo= System.getProperty("java.vendor");
if (tipo.indexOf("Netscape")!=-1)
{
  PrivilegeManager.enablePrivilege("UniversalFileRead");
  PrivilegeManager.enablePrivilege("UniversalFileWrite");
}
```

Para dar soporte a ambos navegadores, ya que uno carga el fichero SimSimplez.cab y otro el SimSimplez.jar, el *applet* se ha declarado en el código HTML de la siguiente forma:

```
<APPLET
  CODE     = "simplez.AppletLanzador.class"
  NAME     = "Applet1"
  WIDTH    = 1
  HEIGHT   = 1
  ALIGN    = middle
  CABBASE  = "SimSimplez.cab"
  ARCHIVE  = "SimSimplez.jar"
>
</APPLET>
```

Donde se incluye la etiqueta CABBASE para Internet Explorer y ARCHIVE para Netscape Navigator. La clase principal de la aplicación hereda de la clase *Applet* y se llama *AppletLanzador*. Este solamente se usa como medio de iniciar la ejecución en el navegador, ya que lo único que realiza es lanzar un objeto de la clase *MainFrame*, que es la ventana principal del simulador, siendo independiente de la ventana del navegador.

4.1.4.2. Versión en formato JAR

Se usa el mismo fichero JAR creado para el navegador de Netscape, como se explica arriba. Al incluir el fichero *manifiest.mf* e indicar en este la clase principal de la aplicación, solamente es necesario disponer del fichero *SimSimplez.jar* y ejecutar:

```
java –jar SimSimplez.jar
```
o
```
javaw –jar SimSimplez.jar
```

para lanzar la aplicación en cualquier plataforma que soporte una versión del JDK o el JRE 1.1 o superiores de Sun. Si se usa la segunda alternativa (*javaw*) se creará un nuevo proceso que no utiliza la consola.

Los usuarios pueden descargar de la web del proyecto los ficheros *SimSimplez.zip* y *SimSimplez.tgz* que contienen los ficheros:

- *SimSimplez.jar*: Las clases Java de la aplicación comprimidas.
- *SimSimplez.bat*: Fichero script BAT para sistema Windows.
- *SimSimplez.sh*: Fichero script shell para sistema Linux.

El fichero BAT para Windows realiza una llamada a *javaw* como se ha descrito arriba, mientras que el script shell la realiza a *java*.

4.1.4.3. Versión ejecutable

Se trata de un fichero ejecutable para plataformas Windows 9X/2000 (.exe) que no precisa de la instalación de un JDK o JRE, al usar la *Microsoft Virtual Machine* directamente. Para su creación se ha usado la herramienta *jexegen* incluida en el mismo paquete de herramientas "Microsoft SDK for Java 4.0" que se ha usado para el firmado del applet para Internet Explorer. Su uso es el siguiente:

```
jexegen /w /out:SimSimplez.exe /main:simplez.AppletLanzador
simplez\*.class componentes\*.class
```

Donde se ve que se especifica la clase principal. El parámetro "/w" indica que la aplicación no usará una consola, sino sólo ventanas.

4.1.5. Rendimiento

La velocidad de la emulación depende del tipo de programa que se emula, aunque existe una diferencia significativa según la versión del simulador usada.
Como valores indicativos se muestran los siguientes valores que se han obtenido simulando el programa "Combinatorios.sim" en un PC con sistema Windows 98 y procesador AMD K7 a 900Mhz:

Versión del simulador	**Ciclos de reloj / segundo (media)**
Ejecutable (.exe)	1869
Ejecutable JAR	5709
Desde navegador I.E.	1774
Desde navegador Netscape	5570

TABLA 3. Velocidad del simulador de Símplez

4.2. Simulador de Algorítmez

El software desarrollado emula el procesador Algorítmez, además de ensamblar programas escritos para el mismo, siguiendo las especificaciones dadas en [1- Gregorio Fernández]. En las siguientes secciones se expone el modelo de este procesador y las decisiones y técnicas usadas a la hora de implementarlo.

4.2.1. Especificación de la máquina Algorítmez

Las características del procesador implementado en este trabajo son las siguientes:

- Arquitectura clásica de von Neumann. El procesador ejecuta las instrucciones de una en una y no existen memorias caché.
- Compartición del mismo espacio de memoria tanto para la memoria de programa como para memoria de datos.
- Juego de 57 instrucciones distintas y 4 modos de direccionamiento básicos distintos.
- Espacio direccionable de memoria de 64kb. En esta implementación, toda la memoria principal es memoria RAM.
- Almacenamiento en memoria con formato extremista menor para palabras de 16 bits: Byte alto en dirección más baja.
- Espacio de entrada / salida separado del espacio de memoria principal. Este tiene una capacidad de direccionamiento de 256 puertos de E/S de 8 bits de ancho cada uno.
- Reconoce una interrupción externa enmascarable, además de otras cuatro internas. En el simulador no se ha implementado la interrupción no enmascarable externa.
- El modelo de interrupciones externas usado es el de vector común.
- La representación numérica usa el convenio de complemento a dos para los números enteros negativos.
- No soporta números de coma flotante, aunque se puede emular por software.
- Dos periféricos: un teclado y una pantalla.
- No se incluye sistema operativo, lo que quiere decir que los programas escritos para este simulador deben manejar por sí mismos la E/S.

4.2.1.1. Modelo de programación

Al nivel de máquina convencional, la implementación usada de Algorítmez contiene los siguientes elementos:

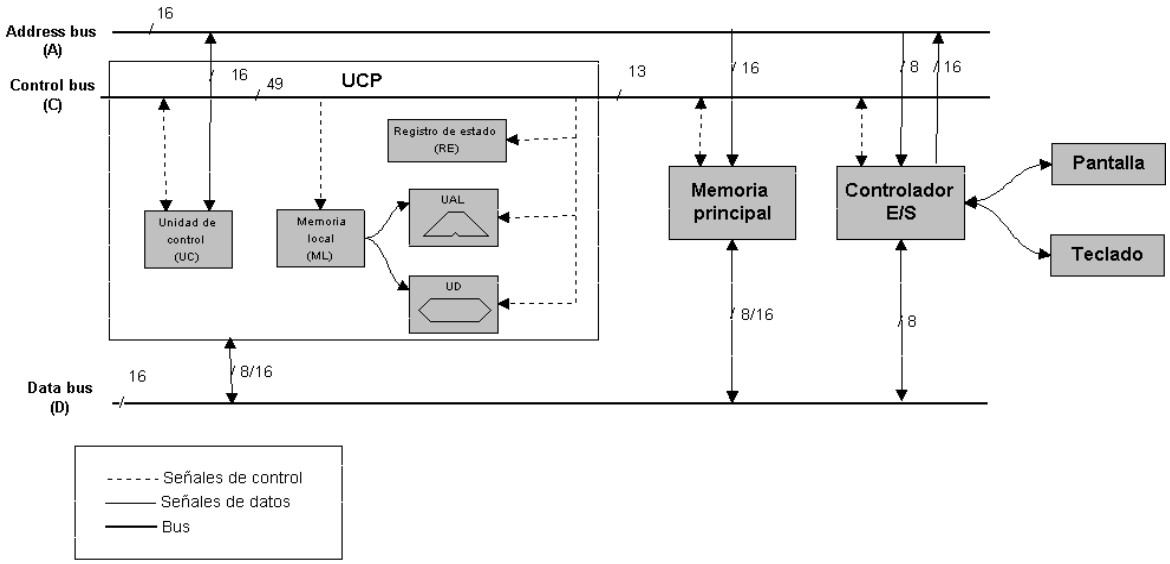


FIGURA 10. Elementos de la implementación de Algorítmez

Estos elementos son sólo los visibles al programador. Existen mas registros y unidades que en este trabajo nos resultarán transparentes, tanto para simulación como para el programador. A continuación se detalla cada uno de los bloques de la UCP (Unidad Central de Proceso):

- UC (Unidad control). Se encarga de coordinar a las demás unidades de la UCP mediante la generación de las señales de control necesarias en cada momento para ejecutar instrucciones continuamente.
- ML (Memoria local). Se compone de 16 registros de 16 bits. Representan una pequeña memoria de acceso inmediato por la UCP, y será sobre la que el programador realizará la mayoría de operaciones. Hay dos registros especialmente importantes. Uno de ellos es el CP (Contador de programa) que se corresponde con el registro número 15 (el último, ya que se numeran desde el registro cero), y el PP (Puntero de pila) que es el registro número 14.

- UAL (Unidad aritmético lógica). Se encarga de realizar operaciones sobre uno o dos operandos. Puede realizar un total de 16 operaciones distintas de naturaleza lógica y aritmética. Opera sobre operandos tanto de 8 como de 16 bits.
- UD (Unidad de desplazamiento). Complementa la UCP con otras 8 operaciones distintas, sobre un solo operando todas ellas. Las operaciones son diversos tipos de desplazamientos y rotaciones. Opera sobre operandos de 16 bits.

- MP (Memoria principal). En el simulador se corresponde con un único bloque continuo de memoria tipo RAM de 64 kb, desde la dirección 0x0000 hasta 0xFFFF.

- RE (Registro estado). Es un registro adicional de 16 bits, aunque solo se usan algunos bits, como se detalla en la siguiente tabla. Los bits no implementados se leen como cero.

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	0	0	0	0	0	SUP	RAS	PIN	0	0	0	H	C	N	V	Z
Valor inicial	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

TABLA 4. Registro de estado

Cada bit actúa de flag o indicador de una determinada acción o suceso. A continuación se detallan sus utilidades:

- Z,V,N,C,H. Son los indicadores lógico-aritméticos que indican que en la última operación de la UAL se ha producido, respectivamente: Cero (zero), desbordamiento (overflow), valor negativo, acarreo (carry) y acarreo parcial (half-carry).
- PIN. Sirve de máscara para permitir (1) o prohibir (0) la entrada de interrupción externa enmascarable.
- SUP. Indica si el procesador trabaja en modo supervisor (1) o en modo usuario (0).
- RAS. Cuando está a 1 activa el modo rastreo. En este modo se genera una interrupción de rastreo con cada instrucción ejecutada, útil para la implementación de depuradores, por ejemplo.

4.2.1.2. Formato de instrucciones

Para que el procesador pueda ejecutar las instrucciones, debe conocer el número de operandos que usa cada una, si se usa o no direccionamiento a memoria, etc. Para hacer esto posible se adopta un formato estándar de instrucción, cuya longitud depende del tipo de instrucción, modos de direccionamiento, etc. El primer byte es estándar y común para todas las instrucciones y en él se indica si se necesita o no el siguiente byte:

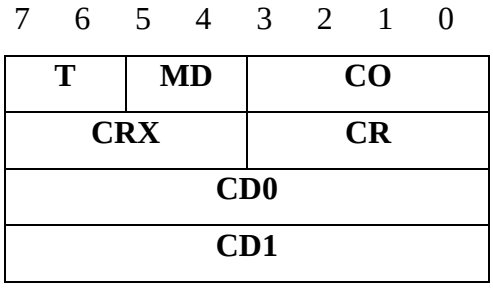


FIGURA 11. Representación de las instrucciones

El campo T, de dos bits, indica el tipo de la instrucción. Se agrupan las instrucciones que usan el mismo número y tipo de operandos. Existen cuatro tipos, el 0,1,2 y 3, y las características de cada uno de ellos son sus operandos:

- Tipo 0: Implícitos. Por ejemplo, las instrucciones que se refieren a bits concretos del registro de estado.
- Tipo 1: Solo registros. Opera con uno o dos registros de la memoria local.
- Tipo 2: Memoria y registros. Opera con una dirección de la memoria principal y con un registro de la memoria local.
- Tipo 3: Solo memoria. Opera con una dirección de la memoria principal.

El campo MD, de dos bits, indica el modo de direccionamiento usado. Junto con el campo CO (opcode) y con el tipo de instrucción, se identifica inequívocamente la instrucción y sus operandos. Los campos CD0 y CD1 sólo se usan para ciertos modos de direccionamiento para almacenar sus operandos.

4.2.1.3. Juego de instrucciones

En [1- Gregorio Fernández] se puede ver la lista detallada de todas las instrucciones y su funcionamiento. Aquí sólo se presentará la lista de los mnemónicos de las instrucciones de cada uno de los cuatro tipos, junto con su *opcode* en hexadecimal:

OPCODE / TIPO	0	1	2	3
$0	CLRC	SHR	ADD	LD.E (*)
$1	CLRV	SHL	ADD.B	ST.E
$2	EI (*)	ROR	ADDC	LD.B.E
$3	DI (*)	ROL	ADDC.B	ST.B.E
$4	BRK	RORC	SUB	CMP
$5	BRKV	ROLC	SUB.B	CMP.B
$6	NOP	SHRA	SUBC	CALL
$7	WAIT (*)	SHLA	SUBC.B	BR
$8	HALT (*)	IN (*)	AND	BC
$9	RET	OUT (*)	OR	BNC
$A	RETI (*)	PUSH	LD	BV
$B	*X*	POP	LD.B	BNV
$C	*X*	CLR	ST	BN
$D	*X*	NOT	ST.B	BNN
$E	*X*	NEG	*X*	BZ
$F	*X*	ADJ	*X*	BNZ

X = *Opcode* no usado

* = Instrucción privilegiada

TABLA 5. Tabla de instrucciones de Algorítmez

NOTA: Si el simulador se encuentra con un opcode incorrecto, lo procesa como un NOP.

4.2.1.4. Interrupciones y periféricos

El simulador de Algorítmez incluye dos periféricos: un teclado y una pantalla, accesibles a través de los puertos de E/S. El teclado usa los puertos 0 y 1, mientras que la pantalla los 2 y 3. El resto de puertos no se usan en el simulador.
Los puertos pares (0 y 2) son de estado, y los impares (1 y 3) de datos. El significado de los bits de los puertos de estado es común:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	X	X	X	X	X	X	INT	PR
Valor inicial	0	0	0	0	0	0	0	0/1

FIGURA 12. Puertos de estado

- X: Bit no usado
- INT: Interrupciones permitidas para este periférico si se activa (1).
- PR: Cuando su valor es 1 significa para el periférico esta preparado: Para el teclado esto significa que una tecla ha sido pulsada, y para la pantalla que está lista para recibir un nuevo carácter.

Cuando se pulsa una tecla, el bit PR del puerto de estado del teclado se activa, produciendo una interrupción si su bit INT esta a 1 y guardando el código ASCII de la tecla pulsada en el puerto de datos del teclado (1). Si se quiere escribir un carácter en la pantalla, se debe esperar a que el bit PR del puerto de estado de la pantalla se active (1) y entonces escribir el código ASCII del carácter en su puerto de datos (3).

Para acceder a ambos periféricos en el simulador, se usa una ventana de terminal de E/S donde se muestra el contenido de la pantalla (50 columnas por 21 filas), un cuadro de entrada de texto (simulando el teclado) y un indicador del contenido del buffer del teclado. Este último es de un solo carácter lo que significa que hasta que el programa emulado de Algorítmez no lea el carácter del puerto de datos del teclado, éste no aceptará mas pulsaciones de teclas.

En cuanto a las interrupciones, el modelo usado para las interrupciones externas es el de vector común: Cuando se detecta una petición de interrupción de uno de los dos periféricos, el bit INT de su puerto de estado está a 1 y el bit PIN de habilitación general de interrupciones del registro de estado está también a 1, se guarda en la pila el CP y el RE, SUP se pone a 1, PIN a 0 y se salta a la dirección indicada por el vector de interrupción externa (Ver tabla más abajo).

Las interrupciones internas no se pueden prohibir con el bit PIN, y son las siguientes:

- VMU (Violación de modo usuario) Es la única interrupción que se produce antes de terminar de ejecutar una instrucción. Indica un intento de ejecución de instrucción privilegiada en modo usuario (SUP=0)
- BK (Break) Interrupción software que se lanza al ejecutar una instrucción BRK.
- BKV (Break on Overflow) Igual que la anterior, pero sólo lanza la interrupción si el bit V del registro de estado se encuentra a 1.
- RS (Rastreo) Disparada tras cada ejecución de instrucción si el bit RAS está a 1.

CAUSA	DIRECCIÓN (DECIMAL)
Teclado	0
Pantalla	2
Rastreo	256
BRKV	258
BRK	260
No enmascarable	262
Violación de modo	264
Vector común	**266**

TABLA 6. Vectores de interrupción implementados

Cada uno de estos vectores deberá contener una dirección absoluta de 16 bits indicando la dirección de la rutina que procesará la interrupción. Hay que resaltar además que los vectores de la pantalla y teclado no son usados directamente por ésta implementación de

Algorítmez, ya que invariablemente saltará al vector común al recibir una petición de interrupción externa, aunque el usuario puede escribir una rutina de consulta de interrupciones que use estos vectores.

Para mantener la compatibilidad con el simulador existente anteriormente de Algorítmez (Jesús Pérez Garcilópez - 1996), se ha introducido el uso de una tabla de periféricos (dirección 512 de memoria), aún cuando no hay sistema operativo. En ésta se guarda en un determinado formato una entrada por cada periférico, indicando si se encuentra operativo y cual es su registro de estado. Esta resulta útil para implementar la rutina de consulta de interrupciones.

4.2.2. Ensamblador

4.2.2.1. Especificación del lenguaje ensamblador

El programa ensamblador se encarga de traducir un programa escrito en lenguaje ensamblador simbólico a código maquina capaz de ser interpretado por la máquina Algorítmez. Cada línea puede contener solo uno de los siguientes tipos de sentencias: pseudoinstrucciones, directivas o instrucciones, y cada uno de sus componentes deben ir separados por al menos un espacio en blanco o un tabulador. En ningún caso es obligatorio el uso de un espacio en blanco como primer carácter de una línea sin etiqueta, ni la colocación en determinadas columnas, aunque esto conllevara algunas restricciones, como se verá más abajo. A continuación se detalla cada uno de los tipos de sentencias que soporta el ensamblador:

- Directivas: Son sentencias que sólo sirven para dar órdenes al programa ensamblador y que no generan contenido alguno en la memoria. El ensamblador de Algorítmez soporta las siguientes:

  - ORG <Constante 16 bits>

    Indica al programa ensamblador a partir de que dirección de la memoria tiene que guardar las instrucciones o pseudoinstrucciones que le siguen.

  - END <Constante 16 bits>

    Indica al programa ensamblador que finaliza el código fuente e indica la dirección del punto de entrada para el programa.

  - <Nombre> EQU <Valor>

    Declara un nuevo símbolo y su valor correspondiente. Siempre que el ensamblador encuentre el nombre del símbolo en una línea de código, lo sustituirá por su valor.

- Pseudoinstrucciones: Son sentencias que aún no siendo instrucciones del lenguaje ensamblador, sí afectan al contenido de la memoria durante el ensamblado:
  - RES <Constante> y RES.B <Constante>

    Reserva un número de bytes igual al indicado, si se trata de RES.B o el doble, si de trata de RES.

  - DATA.B <Constante> [, <Constante> [, <Constante>, ...]]

    Genera directamente en memoria los valores indicados de un byte. Cada constante debe ir separada con una coma. Se pueden usar cadenas de texto encerradas en dobles comillas (“) con lo que se generará la secuencia de bytes en código ASCII correspondiente.

  - DATA <Constante> [, <Constante> [, <Constante>, ...]]

    Igual que DATA.B con la diferencia de que en éste se permiten constantes de dos bytes. Aunque alguna constante se pudiese representar con solo un byte, o sean caracteres, se representarán de igual manera ocupando dos bytes.

- Instrucciones: Son los mnemónicos que representan las correspondientes operaciones máquina del microprocesador. Cada línea de programa puede contener una sola instrucción. El formato general de una línea de programa con una instrucción es:

TIPO 0: [Etiqueta] <mnemónico>
TIPO 1: [Etiqueta] <mnemónico> .R [,.RX]
TIPO 2: [Etiqueta] <mnemónico> .R, <mem>
TIPO 3: [Etiqueta] <mnemónico> <mem>

Donde:

- Corchetes ([ ]) indican que no siempre es obligatorio.
- .R o .RX son registros. P.ej: .0, .15 , etc...
- <mem> representa alguno de los 8 modos de direccionamiento a memoria de que dispone Algorítmez. A continuación se presenta la sintaxis que debe tener cada uno:

Modo	Sintaxis	Comentarios
Autoincremento	`[.RX++]`	La dirección efectiva es el contenido del registro, antes de incrementarlo.
Indexado	`/<8BITS>[.RX]`	Desplazamiento de 8 bits con signo respecto a un registro.
Autoincremento Indirecto	`[[.RX++]]`	La dirección eficaz es el valor de 16 bits almacenado en la dirección de MP indicada por RX.
Indexado indirecto	`[/<8BITS>[.RX]]`	Desplazamiento de 8 bits con signo respecto a un registro e indirección.
Inmediato	`#<8 o 16 BITS>`	Valor inmediato. Longitud depende de la instrucción.
Relativo a programa	`<16BITS>` ó `$<16BITS>`	La dirección de 16 bits indicada debe estar en el rango: CP-127 a CP+128
Directo	`/<16BITS>`	Dirección de 16 bits absoluta.
Relativo a programa e indirecto	`[<16BITS>]` ó `[$<16BITS>]`	La dirección de 16 bits indicada debe estar en el rango: CP-127 a CP+128

TABLA 7. Modos de direccionamiento de Algorítmez

- La etiqueta (optativa) asociará un nombre a la dirección en memoria, facilitando su referenciación. Las etiquetas también se pueden usar con las seudo-instrucciones, siendo la dirección asociada a la etiqueta en cualquier caso la dirección del contador de direcciones del ensamblador antes de procesar la línea.

- Comentarios: Independientemente del contenido de una línea, se pueden añadir comentarios usando el carácter punto y coma (“;”), con lo que el ensamblador ignorará el contenido del resto de la línea.

4.2.2.2. Restricciones en nombres de etiquetas y símbolos

El ensamblador distingue automáticamente, como se verá en el siguiente punto, si en una línea hay o no etiqueta. Con esto se evita la necesidad de insertar espacios cuando no hay etiquetas. A cambio de esto, no se permite usar etiquetas con nombre igual al de algún mnemónico. En principio la longitud de los nombres de las etiquetas no está limitada.

4.2.2.3. Expresiones numéricas

En los ficheros fuente, cualquier valor numérico se considerará por defecto en decimal, pero se pueden usar los siguientes prefijos para cambiar la base:

Prefijo	Base
B’	Binario
Q’	Octal
D’	Decimal
H’	Hexadecimal

TABLA 8. Prefijos numéricos en Algorítmez

4.2.2.4. Funcionamiento del ensamblador

Tanto el ensamblador como el simulador de Algorítmez se han creado en una misma clase Java, llamada “CMVAlgoritmez”. Su interfaz externa se compone principalmente por los siguientes métodos:

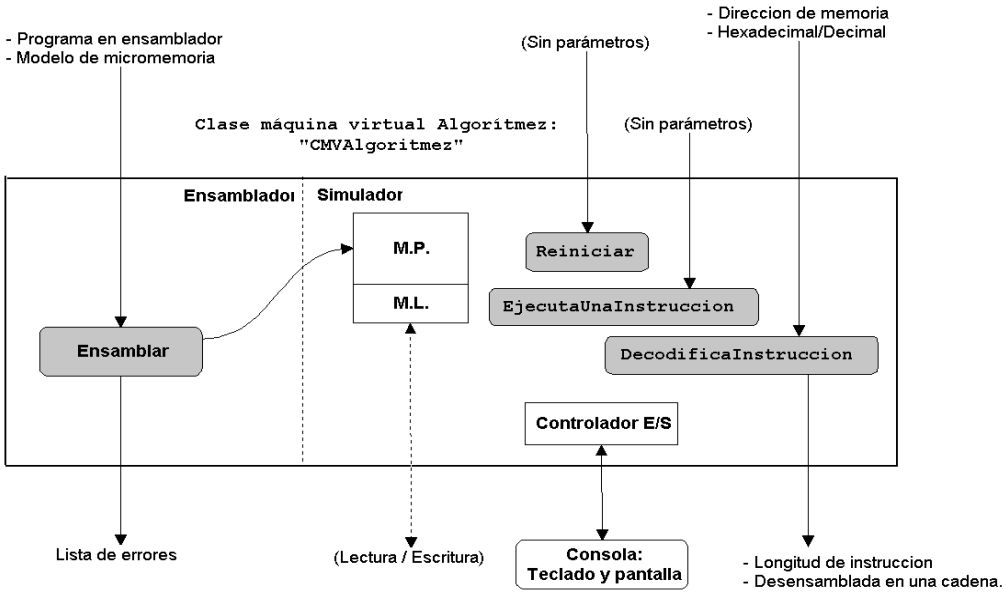


FIGURA 13. Interfaz resumida de la clase “CMVAlgoritmez”

En este punto se va a analizar solamente la parte del ensamblador, que como se ve, parte de una serie de líneas con un programa fuente en Algorítmez, para producir directamente en la memoria del simulador el programa en código máquina correspondiente.
Aunque un ensamblador típico suele generar un fichero de código objeto, en este caso teniendo en cuenta la tecnología que se ha elegido (Java y Applets) se ha optado por eliminar este paso, para que si el usuario elige por cualquier razón no aceptar la petición del simulador de acceso a ficheros, el simulador al menos siga funcionando correctamente, con la lógica limitación de tener que escribir manualmente los programas.

Para ensamblar un programa es necesario recorrer cada una de las líneas del programa, analizando los modos de direccionamiento empleados, sustituyendo los valores de los símbolos y de las etiquetas por sus respectivos valores, buscando el *opcode* para cada mnemónico, etc. y con estos datos rellenar la estructura de formato de instrucción descrita en 4.2.1.2. que será guardada en memoria. Para implementar esto ha sido necesario usar los siguientes elementos:

- Contador de posición actual. Indica la posición de memoria en donde se guardarán los datos generados al ensamblar. Este contador se incrementa automáticamente cada vez que se guardan datos en memoria, salvo cuando se trata de la directiva *ORG*, que cambia el valor del contador al valor indicado. Al comenzar el ensamblado vale cero.
- Tabla de instrucciones. Contiene la siguiente información sobre cada instrucción: mnemónico, *opcode*, tipo, acceso a registro y acceso a memoria. Estos dos últimos son valores lógicos que indican si la instrucción usa como operandos registros y una dirección de memoria, respectivamente. Aunque puede parecer que hay información redundante al añadir estos dos últimos campos, ya que el tipo de la instrucción identifica el tipo de operandos, existe un problema con dos instrucciones: *CMP* y *CMP.B*, que siendo de tipo 3 (operandos sólo memoria) usa un registro y una dirección de memoria (el correspondiente al tipo 2). Debido a esto es necesario guardar por un lado el tipo de instrucción y por otro el tipo de operandos. Aunque se podría haber modificado parte del ensamblador para tener en cuenta estas dos instrucciones especiales, se ha optado hacerlo de esta manera para conseguir mayor

generalidad en el ensamblador. Para más detalles del porqué de estas dos instrucciones, véase el capítulo 13 de [1- Gregorio Fernández].

- Dirección inicial: Es la variable donde se almacena el punto de entrada al programa. Esta viene especificada en el operando de la directiva *END*.
- Lista de símbolos: Contiene pares (nombre, valor) donde ambos son cadenas de texto. En esta lista se almacenan los símbolos declarados con las directivas *EQU*. El valor puede ser tanto un valor numérico como texto.
- Lista de etiquetas: Contiene pares (nombre, valor) donde el primero es una cadena y el segundo un valor numérico, correspondiente a la dirección donde están colocadas las etiquetas.

El problema a resolver más evidente del ensamblador es la utilización de etiquetas que aún no están definidas. Para solucionarlo es necesario dividir el proceso de ensamblado en dos pasadas. En cada una de las pasadas, se recorre completamente el código línea por línea, pero en cada ocasión se realizan tareas diferentes:

- 1ª pasada: Aún se desconocen los valores de las etiquetas declaradas después de su uso. Sin embargo, y debido al formato de instrucciones de Algorítmez que es de longitud variables, se intenta ensamblar cada instrucción y analizar su modo de direccionamiento aún ignorando el valor de la etiqueta (se usa un valor de cero para las desconocidas), ya que es necesario saber cuantos bytes ocupan todas las instrucciones precedentes a una dada para calcular su posición correcta. Además, sólo en esta primera pasada se procesan las directivas *EQU*.

- 2ª pasada: En ésta ya se conocen los valores correctos de todas las etiquetas. Por lo tanto ya se genera el código máquina definitivo y se almacena en las posiciones correspondientes de la memoria del simulador. En esta pasada se ignoran las directivas EQU por ya conocerlas todas.

El mecanismo usado para distinguir si en una línea existe o no una etiqueta es el siguiente:

1. Preprocesar la línea. Esto se realiza siempre con todas las líneas del programa antes de intentarla ensamblar, ya sea en la primera o segunda pasada. En este proceso se

sustituyen los símbolos declarados con EQU por sus valores, se sustituyen los símbolos tabuladores por espacios y se limpian los espacios innecesarios, dejando solamente un espacio entre palabra y palabra, facilitando su análisis.

2. Se divide la línea en palabras (secuencias de caracteres separados por espacios)
3. Se comprueba si la primera palabra puede ser interpretada como una instrucción o pseudoinstrucción. Si es así se procesa y la línea no usa etiqueta.
4. De no ser así, se comprueba si la segunda palabra puede ser interpretada como tal. Si es así, la primera palabra se interpreta como una etiqueta.

4.2.3. Simulador

4.2.3.1. Funcionamiento del simulador

Para la simulación, el simulador guarda en variables los siguientes datos:

- Memoria: Un array de bytes de 64kb de tamaño.
- Registros: Un array de 16 palabras de 16 bits. Aquí se incluyen el puntero de pila y el contador de programa.
- Registro de estado. Una palabra de 16 bits.
- Estado de periféricos: En este caso solo pantalla y teclado.
- Lista de puntos de ruptura: Para la depuración de programas.

El simulador de Algorítmez en sí es sencillo, ya que se ha emulado al nivel de instrucción, con lo que la parte más laboriosa de la implementación es escribir el código que emule a cada una de las distintas instrucciones. La unidad mínima de tiempo al simular es una instrucción, lo que hace que las operaciones a realizar en cada paso sean sencillas:

1. Cargar un byte de la dirección de memoria que indica CP e incrementar este.
2. Analizar este primer byte de instrucción, y si es necesario, leer hasta otros tres bytes.
3. Si es necesario, calcular la dirección efectiva del modo de direccionamiento.
4. Según sea el *opcode* y el tipo, ejecutar las acciones que indica la instrucción.
5. Si es aplicable, actualizar los bits convenientes del registro de estado con el resultado de la operación.
6. Comprobar si hay petición de interrupciones, o se ha producido una interna. Si es así, poner en CP el valor del vector correspondiente para que se continúe en el siguiente ciclo.

Aunque el funcionamiento básico es paso a paso, en la aplicación se permite la ejecución de forma continua. Esto se realiza creando un hilo de ejecución aparte del hilo que maneja el interfaz de usuario, ejecutando continuamente instrucciones, y actualizando periódicamente en la pantalla el estado de la memoria, registros, etc.

Para facilitar el depurado de programas, se ha añadido la capacidad de insertar *breakpoints*, o puntos de ruptura, de forma que si se está ejecutando de forma continua, se interrumpa al llegar a uno de estos.

La emulación de los periféricos se ha realizado de la siguiente forma:

- Teclado: En el constructor de la clase CMVAlgoritmez, hay que pasarle un objeto de tipo *TextComponent*. A este componente se ha añade un *KeyListener*, que en Java es un procesador de eventos del componente, orientado a eventos de teclado. De esta forma, al pulsar una tecla se llama automáticamente a un método definido dentro de la clase emuladora de Algorítmez, que se encarga de actualizar los valores de los puertos si es necesario.

- Pantalla: Al constructor de la clase CMVAlgoritmez también se le pasa un objeto que implemente la interfaz InterfazPantalla, que se define así:

```
interface InterfazPantalla
{
  char      getContenidoPantalla(int col,int row);
  void      setContenidoPantalla(int col,int row,char c);
  int       getCursorX();
  int       getCursorY();
  void      setCursorX(int x);
  void      setCursorY(int y);
  boolean   getListo();
  void      setOcupado();

  void      ActualizaPantalla();
}
```

  Que es usada para comunicarse con el cuadro de diálogo “Consola de E/S” para actualizar el contenido de la pantalla y redibujarla siempre que sea necesario por operaciones del programa de Algorítmez sobre los puertos correspondientes.

En el apéndice A6.2 se dan más detalles sobre la implementación de la aplicación.

4.2.3.2. Puertos y tabla de periféricos

Estos son los puertos implementados en el simulador de Algorítmez.

Puerto	Periférico	Uso
0	Teclado	P. de estado. Bit 0: Preparado. Bit 1: Permitir interrupciones
1		Entrada: Carácter de tecla pulsada.
2	Pantalla	P. de estado. Bit 0: Preparado. Bit 1: Permitir interrupciones
3		Salida: Carácter a imprimir.

TABLA 9: Los puertos de Algorítmez en el simulador.

Cualquier salida a un puerto no definido será almacenada, en ese puerto, aunque no tendrá más efectos. Así como lecturas a puertos distintos de estos, devolverán cero, o el valor que anteriormente se envió, pero no tendrá tampoco más efectos.

Estos puertos siguen las especificaciones dadas en [1-Gregorio Fernández]. A continuación se ven en más detalle el uso de cada puerto, el significado de cada bit y si el puerto es de entrada y salida (E/S), solo entrada (E) o solo salida (S).

Puerto[0]: Teclado – Estado (E/S)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	X						INT	PR
Valor inicial	0						0	0

Solo se puede escribir el bit INT, ignorando la escritura en los demás bits.

- PR: Bit de preparado. Cuando esta a 1, significa que se ha pulsado una tecla y esta lista para ser leída.
- INT: Máscara de interrupciones. Si esta a 0 inhibe la petición de interrupción de éste dispositivo cuando su estado sea de preparado.

Puerto[1]: Teclado – Datos (E)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	CARÁCTER							
Valor inicial	0x00							

Leyendo este puerto se obtiene el código ASCII del carácter que se ha pulsado en el teclado. Ver las limitaciones respecto a los caracteres leídos desde el teclado en el punto 4.3.5.4.

Al realizar una operación de lectura de éste puerto, se pone a cero automáticamente el bit PR del puerto de estado del teclado.

Puerto[2]: Pantalla – Estado (E/S)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	X						INT	PR
Valor inicial	0						0	1

Solo se puede escribir el bit INT, ignorando la escritura en los demás bits.

- PR: Bit de preparado. Cuando esta a 1, significa que el periférico esta preparado para enviar un nuevo carácter a la pantalla.
- INT: Máscara de interrupciones. Si esta a 0 inhibe la petición de interrupción de éste dispositivo cuando su estado sea de preparado.

Puerto[3]: Pantalla – Datos (S)

Bit	7	6	5	4	3	2	1	0
Significado	CARÁCTER							
Valor inicial	0x00							

Al escribir en este puerto, se envía el byte como carácter a imprimir a la pantalla virtual de Algorítmez. Tras hacer esto se pone a cero el bit de preparado del puerto de estado de la pantalla, que se volverá a poner a uno tras un período de tiempo, que se ha elegido de 50 instrucciones ejecutadas. Toda salida a este puerto durante ese periodo de periférico ocupado, será ignorada.

Además, el ensamblador genera automáticamente en memoria una tabla de periféricos en la dirección 512 (H'200). Esta tabla consta de varios campos, aunque sólo se rellenan los primeros. Su contenido se muestra en la siguiente tabla, donde cada fila es un registro de 20 bytes, y sirve para manejar el estado de un periférico. El número entre paréntesis de cada campo indica el número de bytes que ocupa:

Fila	Dir E/S (1)	Estado (1)	Zona E/S (2)	Marco (1)	DF (1)	NBYP (2)	Zona user(2)	Dir aviso(2)	NBL (2)	NBYT (2)	PUNTES (2)	PUNTZU (2)
0	2	0x80	0	0	0	0	0	0	0	0	0	0
1	0	0x80	0	0	0	0	0	0	0	0	0	0
2	0xFF	0	0	0	0	0	0	0	0	0	0	0

TABLA 10: La tabla de periféricos en el simulador de Algorítmez.

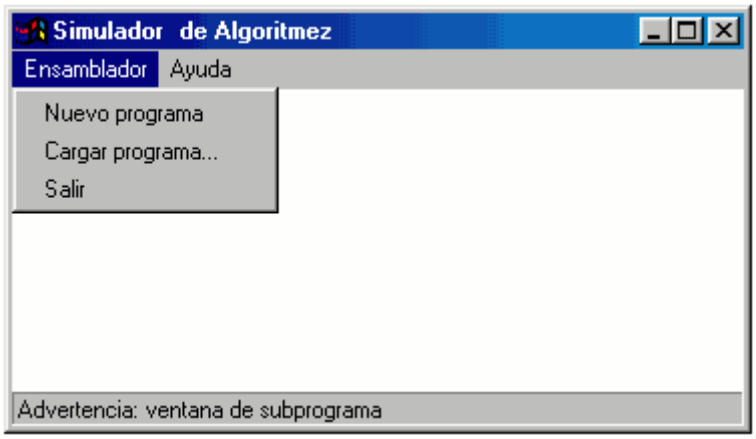
Donde como se ve el primer registro corresponde a la pantalla (puerto 2) y el segundo al teclado (puerto 0), mientras que el final de la tabla se indica con un valor H'FF en el campo de dirección de puerto de estado.

Esta rutina es útil a la hora de implementar una rutina de consulta de interrupciones que procese la interrupción de vector común para averiguar el periférico fuente de la interrupción.

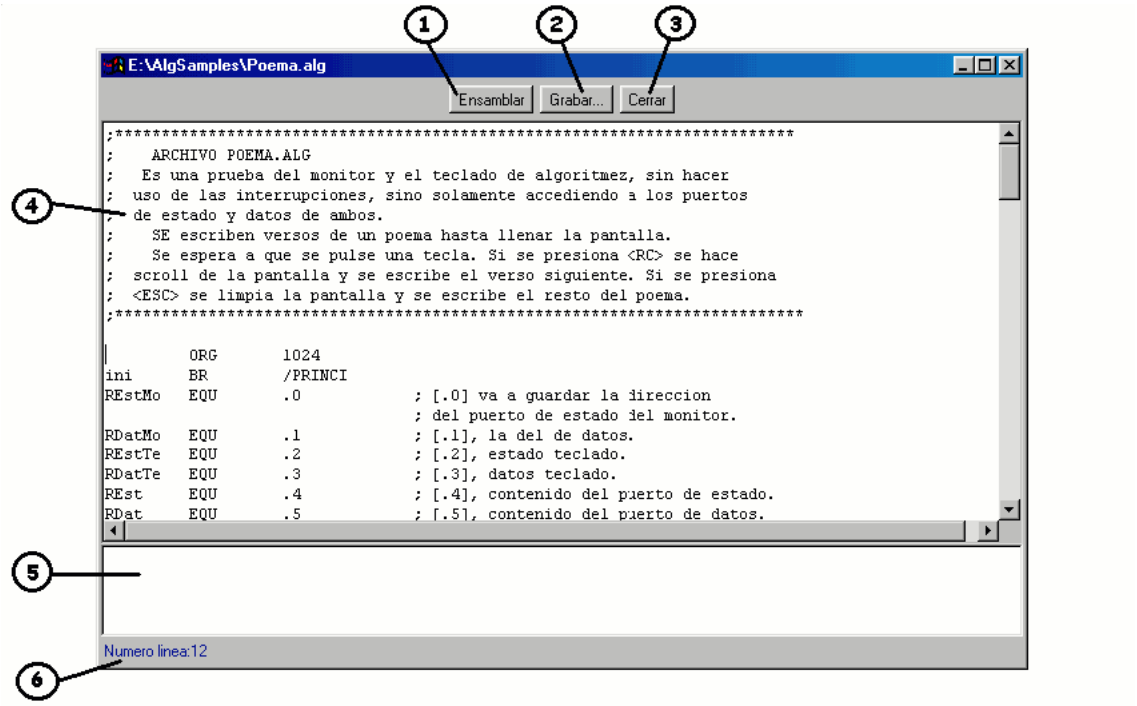
4.2.4. Interfaz de usuario

Aprovechando el diseño orientado a objetos, se ha realizado de forma que pueda haber tantos programas de Algorítmez (en edición o en simulación) como se quiera en cada momento, cada uno independiente del resto (periféricos, memoria, etc...)

Al usar Java y para aprovechar la facilidad de distribución de la aplicación desde un servidor web, se debe implementar un *applet* en una página HTML. Pero debido a las molestias que puede acarrear tener la aplicación dentro de la ventana de los navegadores, se ha elegido usar únicamente un *applet* como lanzador de la ventana principal del programa, independiente al navegador:



Desde esta ventana se puede abrir el editor de programa en ensamblador, y si el usuario ha dado su consentimiento al navegador para dar privilegios de acceso a ficheros a la aplicación, cargar el código fuente de los programas:



La ventana del editor contiene los siguientes elementos:

1- **Botón "ensamblar":** Ensambla el programa escrito en el editor. Si éste se cargó de un fichero y se ha modificado, se usa la versión escrita, sin actualizar el fichero de forma automática. Si el programa no contiene errores, se cargará automáticamente el simulador y se abrirá la ventana de simulación.

2- **Botón "grabar":** Guarda en un fichero el programa. Para que funcione correctamente, el usuario debe haber aceptado los permisos del *applet*.

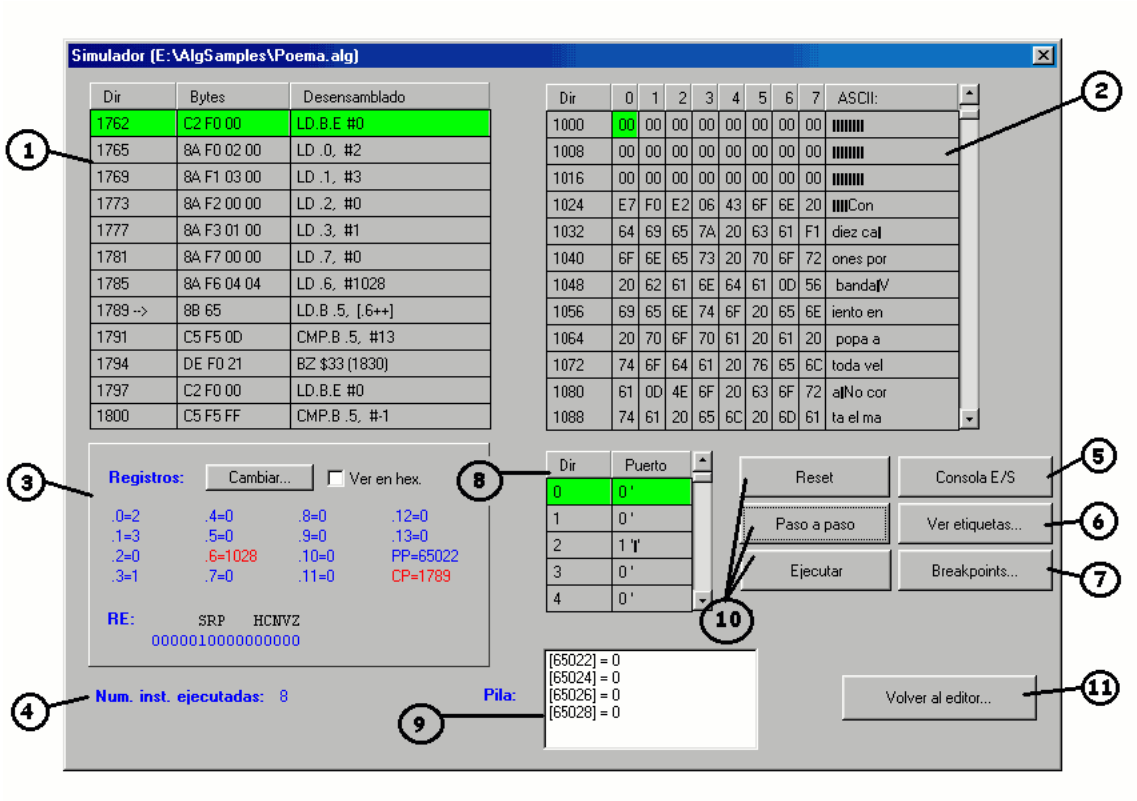
3- **Botón "cerrar":** Cierra el editor. No guarda automáticamente el programa a un fichero.

4- **Texto**: El código del programa en sí.

5- **Ventana de mensajes:** Aquí es donde se muestran los mensajes del ensamblador, así como los posibles errores en el código. Haciendo doble clic sobre un error, se marcará automáticamente éste en el texto. Esta ventana aparece automáticamente al ensamblar. Si se desea cerrarla o abrirla manualmente, hay que pulsar el botón derecho sobre el texto del programa y elegir "Ocultar/mostrar mensajes".

6- **Contador de línea:** Aquí se muestra la línea del programa sobre la que está situado el cursor.

La ventana principal del simulador es ésta:



1- **Tabla de desensamblado:** Aquí se muestran unas pocas instrucciones desensambladas. Por defecto, se muestra la parte a donde apunta el CP, pero se puede ver la zona deseada de la memoria pulsando el botón derecho sobre la tabla y eligiéndolo. La siguiente instrucción a ejecutar, indicada por CP, se muestra con un "→" a la izquierda de esta tabla.

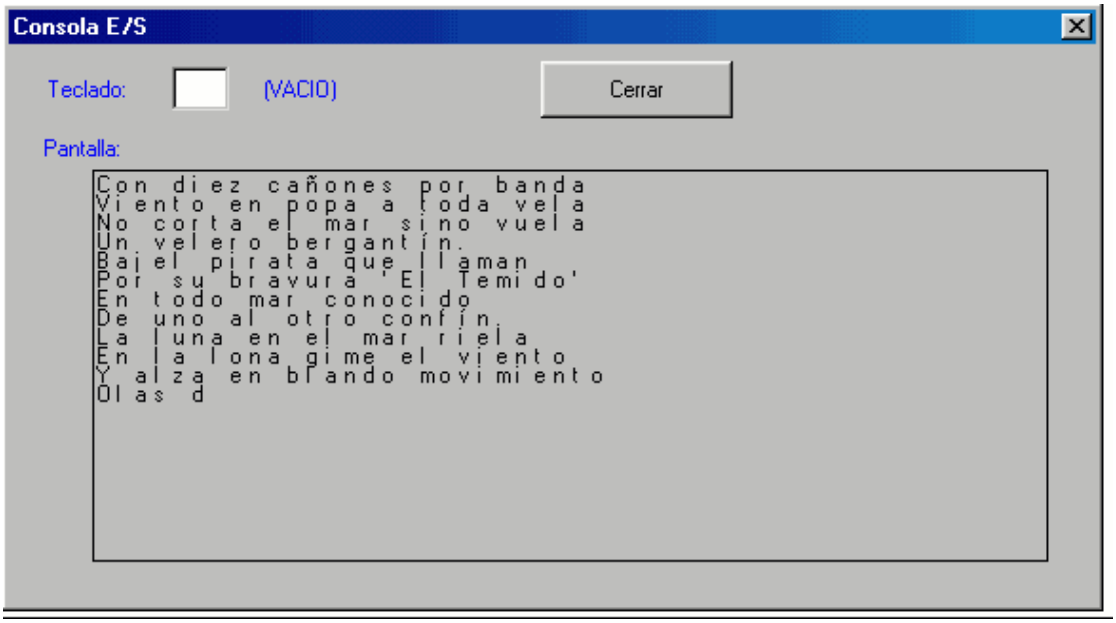
2- **Tabla de dump:** En esta tabla se muestra el contenido de toda la memoria de Algorítmez, los 64Kb. Para modificar un valor o ir a una dirección determinada, pulsar el botón derecho sobre la tabla y elegir la operación en el menú.

3- **Registros:** Aquí se muestra el contenido de los 16 registros de la ML de Algorítmez. Además se muestra el contenido del registro de estado (RE) bit a bit, indicando el significado de cada uno. Se muestran en rojo los registros o bits

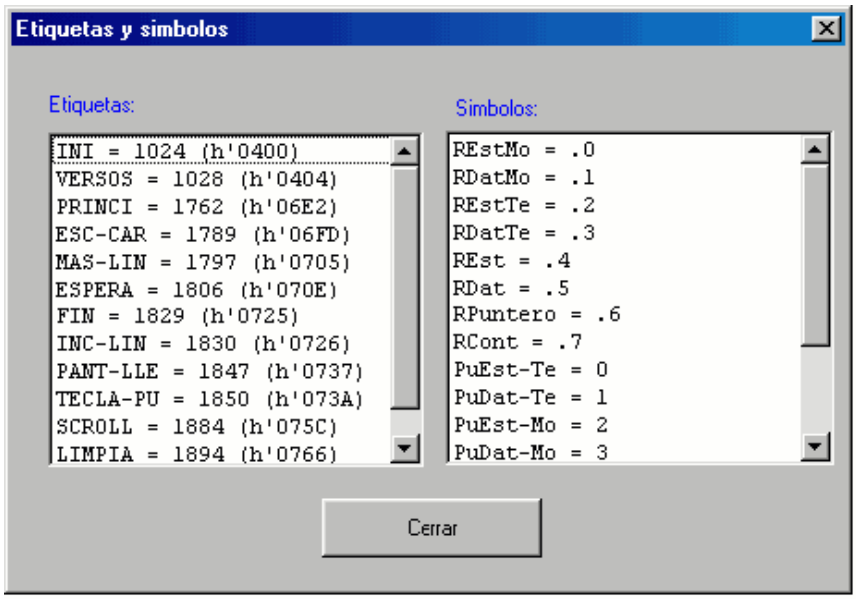
que han sido modificados por la ejecución de la última instrucción, lo que facilita el seguimiento del programa.

4- **Contador:** Indica el número de instrucciones ejecutadas desde el reset de la máquina virtual Algorítmez.

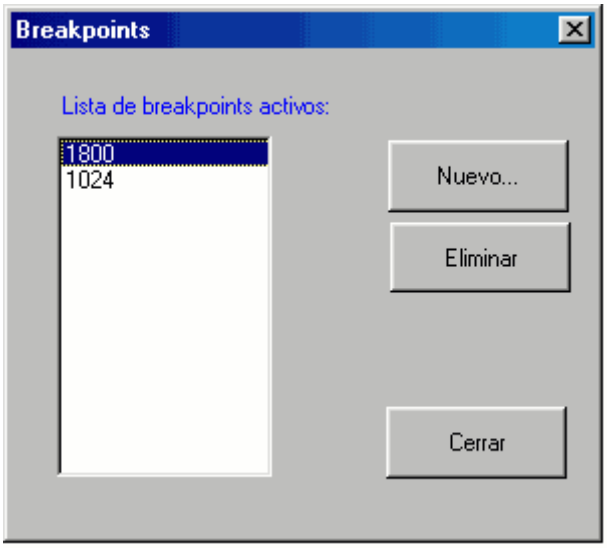
5- **Ver consola E/S:** Muestra la ventana de la consola, donde estan el teclado y la pantalla de Algorítmez:



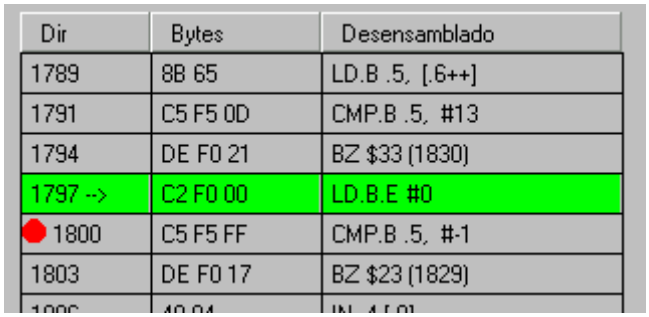
6- **Ver etiquetas:** Muestra las etiquetas y símbolos encontrados durante el ensamblado, así como sus valores:



7- **Breakpoints:** Muestra la ventana de manejo de *breakpoints*, donde se pueden ver todos los *breakpoints* colocados hasta el momento, añadir nuevos o borrarlos:



Otra forma más rápida de poner o quitar un *breakpoint* es en el menú de la tabla de desensamblado, al pulsar el botón derecho y pulsar "Establecer / quitar breakpoint" en el menú contextual. Un *breakpoint* se representa en el desensamblado como un punto rojo a la izquierda:



Dir	Bytes	Desensamblado
1789	8B 65	LD.B .5, [.6++]
1791	C5 F5 0D	CMP.B .5, #13
1794	DE F0 21	BZ $33 (1830)
1797 -->	C2 F0 00	LD.B.E #0
1800	C5 F5 FF	CMP.B .5, #-1
1803	DE F0 17	BZ $23 (1829)

8- **Puertos:** En esta lista se muestra el contenido de los 256 puertos de Algorítmez, aunque sólo se usan los 4 primeros para la pantalla y el teclado. El resto se leen como cero.

9- **Pila:** Aquí se muestran los valores de la pila, así como sus direcciones en la memoria. Por ejemplo, tras una interrupción, se verá el contenido del CP y del RE.

10- **Botones de control de ejecución:** El botón “Reset” en cualquier momento reinicia la máquina virtual (MV), vuelve a cargar en la memoria el contenido del programa ensamblado y coloca en CP la posición de entrada al programa.
    El botón “Paso a paso” ejecuta una sola instrucción. El botón “Ejecutar” activa el modo de ejecución continuo, hasta que se pare manualmente, se encuentre un breakpoint o una instrucción HALT.
11- **Salir del simulador.** Cierra el simulador, volviendo al editor del programa.

4.2.5. Detalles de la implementación en Java

Tanto el ensamblador como el simulador se han creado dentro de una misma clase de Java "CMVAlgoritmez" para facilitar la reutilización del código.

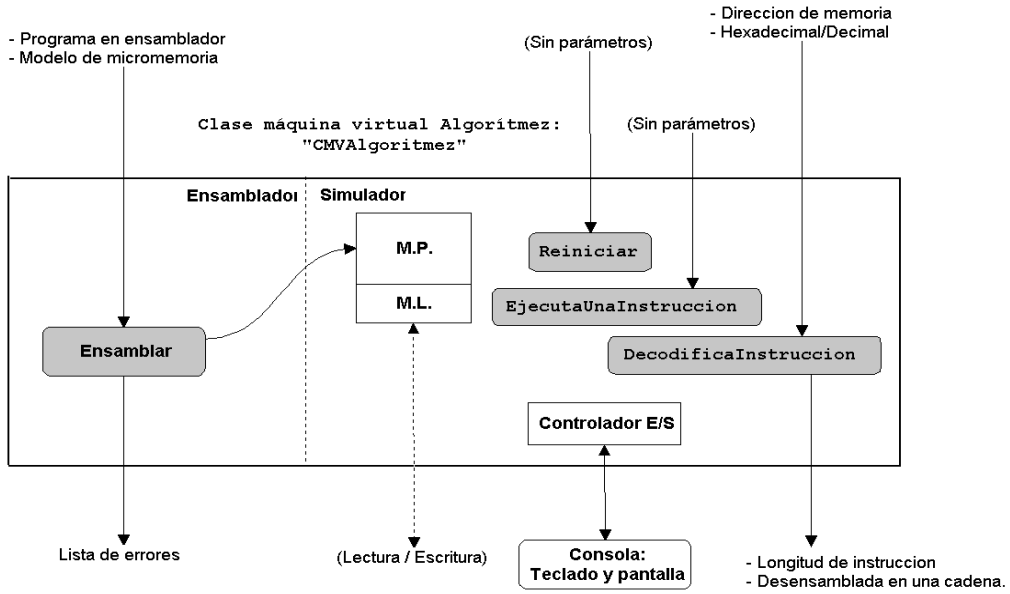


FIGURA 14. Interfaz resumida de la clase "CMVAlgoritmez"

La interfaz que ésta clase ofrece se ha creado lo más sencilla posible, de forma que los principales métodos públicos (aparte del constructor) son estos:

- *Reiniciar().* Reinicia la máquina virtual, sus registros, periféricos.
- *Ensamblar(String ASM,Vector ListaErrores).* Ensambla el programa que se le pasa, devolviendo una lista con los posible errores.
- *EjecutaUnaInstruccion().* Ejecuta una sola instrucción y si es el caso, procesa una petición de interrupción, saltando al vector necesario.
- *DecodificaInstruccion(int Dir, bolean EnHex):* Que devuelve un Vector conteniendo la longitud en bytes de la instrucción que se encuentra en la posición "Dir" de la memoria, y una cadena representándola desensamblada.

Internamente a esta clase, se han creado otras tres:

- CEtiqueta
- CSimbolos
- CInstruccion

De las cuales, las dos primeras son sólo contenedoras de los pares nombre-valor de las etiquetas y los símbolos, respectivamente. La clase CInstruccion guarda información sobre una instrucción Algorítmez, pero además se le ha asignado parte del código del ensamblador: la que procesa los operandos y los guarda en los respectivos campos de la instrucción y la que a partir de todos los datos de una instrucción, genera la lista de bytes a insertar en memoria.

Para comunicarse con el resto de la aplicación, se han definido *interfaces*. Por ejemplo, para avisar a las tablas que muestran el contenido de la memoria que ésta ha cambiado y hay q actualizarlas, se ha definido la siguiente interfaz:

```
interface InterfaceVMRepaint {
  void OnVMRepaint();
}
```

Mientras que para comunicarse con la ventana que muestra la consola de Algorítmez, se ha definido esta otra:

```
interface InterfazPantalla {
  char      getContenidoPantalla(int col,int row);
  void      setContenidoPantalla(int col,int row,char c);
  int       getCursorX();
  int       getCursorY();
  void      setCursorX(int x);
  void      setCursorY(int y);
  boolean   getListo();
  void      setOcupado();

  void      ActualizaPantalla();
}
```

De esta forma se consigue modularidad en el diseño, y se puede usar el CMVAlgoritmez en cualquier otra aplicación mientras se cumplan estas interfaces.

4.2.6. Distribuciones

En el simulador de Algorítmez se han creado también tres distribuciones de la aplicación:

- Ejecutable desde web a través de un applet y un navegador.
- Versión ejecutable .exe para Windows.
- Versión JAR para ejecución multiplataforma.

Los detalles de la creación de las distintas distribuciones coinciden con los pasos seguidos para el simulador de Símplez y explicados en el punto 4.1.4.

4.2.7. Rendimiento

La velocidad de la emulación depende del tipo de programa que se emula, aunque existen diferencias significativas según la versión del simulador usada.

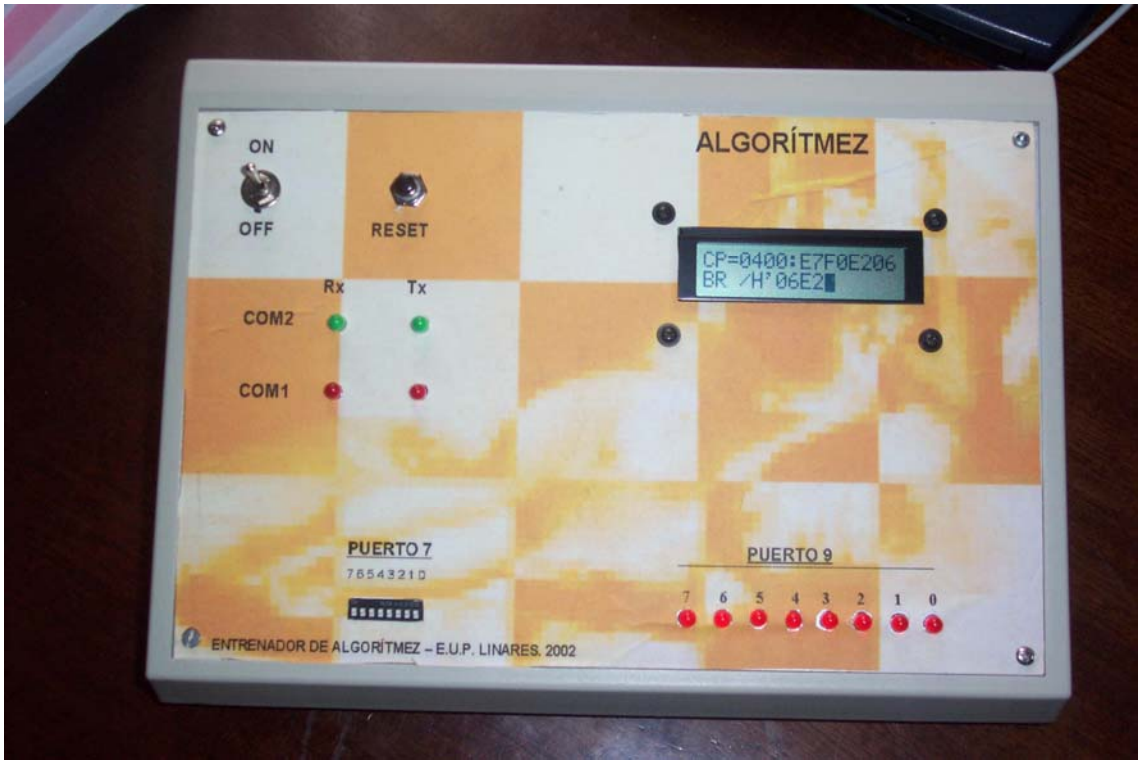
Como valores indicativos aproximados se muestran los siguientes valores que se han obtenido simulando un programa Algorítmez que muestra un texto largo por pantalla, en un PC con sistema Windows 98 y procesador AMD K7 a 900Mhz:

Versión del simulador	**Instrucciones / segundo (media)**
Ejecutable (.exe)	6846
Ejecutable JAR	5134
Desde navegador I.E.	7663
Desde navegador Netscape	8647

TABLA 11. Velocidad del simulador de Algorítmez

Como se ve el número de instrucciones por segundo conseguidas en este simulador es superior incluso al número de ciclos de reloj emulados por segundo en el emulador de Símplez, cuando se podría suponer lo contrario ya que este microprocesador es mucho más complejo.
Este resultado es debido a la mayor complejidad del simulador de Símplez (no al mismo Símplez), ya que se tiene que tratar con todas las señales y buses individualmente, mientras que en el de Algorítmez directamente se emulan las consecuencias de la ejecución de una instrucción, no todos los mecanismos internos que los producen como en Símplez.

4.3. Entrenador hardware de Algorítmez



4.3.1. Introducción

Se ha desarrollado un sistema capaz de emular la ejecución de programas completos Algorítmez sobre una plataforma hardware basada en microcontrolador, simulando una máquina virtual Algorítmez (MVA en adelante). Esto permite dar un uso práctico a los programas Algorítmez, además de incluir amplias funciones para usar el entrenador como depurador.

Usando el programa "Comunicador" en el PC, se pueden ensamblar programas Algorítmez, enviarlos vía puerto serie al entrenador y allí ejecutarlos, depurarlos paso a paso viendo en cada momento la instrucción desensamblada, modificar el valor de los registros, etc...

Para el usuario, el entrenador presenta las siguientes interfaces y periféricos:

- Una pantalla LCD, donde se muestran los mensajes del entrenador, menús, etc... También se usa para mostrar el contenido de la pantalla de la máquina virtual de Algorítmez.
- Teclado PS-2. Igualmente, se usa para seleccionar las acciones a realizar por el entrenador, introducir valores, etc... y como teclado de entrada a Algorítmez.
- Indicador de 8 leds y 8 microswitchs, correspondiendo a dos nuevos puertos de Algorítmez.
- Dos puertos serie, COM1 y COM2. El puerto COM1 es usado exclusivamente para comunicar el sistema entrenador con el programa comunicador en el PC, mientras que el COM2 es un puerto serie accesible al programador a través de unos nuevos puertos en la máquina virtual Algorítmez.

Se han desarrollado dos modos de funcionamiento principales para el entrenador: Modo depuración y modo de solo ejecución. Esto se explica mas detenidamente en el punto 4.3.5.1.

4.3.2. Diferencias de la máquina Algorítmez

Aunque la máquina Algorítmez emulada por el entrenador se ha diseñado lo más parecida posible a la especificación dada en [1- Gregorio Fernández] (la usada para los simuladores software) hay unas pequeñas variaciones que se ven a continuación. El resto de las especificaciones se mantienen (Instrucciones, mecanismo y vectores de interrupciones, etc...).

4.3.2.1. Memoria

Se ha querido permitir que los programas almacenados en el entrenador se mantengan en memoria aún cuando se desconecte la alimentación de éste, por lo que se ha desechado la posibilidad de usar memoria RAM externa al microcontrolador, frente a un chip de memoria EEPROM accesible por bus I2C de dos líneas serie. La desventaja de usar este sistema es el retardo de acceso a memoria (algunos microsegundos en lectura, pocos milisegundos en escritura, frente a los pocos nanosegundos de una RAM estática), pero tiene las ventajas que conlleva el permitir almacenar programas aunque se desconecte la alimentación y además simplifica enormemente el diseño, al requerir solamente un bus de dos líneas. Las memorias que se han usado son del fabricante Atmel, y pertenecen a la familia AT24C512, memorias E2PROM I2C de 64Kb (512 Kbits).

Independientemente del tipo de memoria, existe el siguiente problema: Si se le envía desde el PC al entrenador una imagen de 64 Kbytes para que emule un programa, tras la ejecución del programa la memoria se habrá modificado, como consecuencia del uso de variables en RAM, etc... Tras esto, al reiniciar el entrenador la imagen de la memoria no es la original.

Se pueden pensar dos soluciones a este problema:

1. **Ignorarlo**. De esta forma, el programador debe saber de antemano que el contenido de las posiciones que usa como datos o variables, deben ser inicializadas siempre al inicio de cada programa. Esto es un problema

añadido en casos donde se modifiquen los valores de una tabla en memoria, por ejemplo.

2. **Duplicar el banco de memoria**. Si el espacio de memoria de Algorítmez es de 64 Kbytes, se usa un segundo banco de otros 64 Kbytes a forma de "copia de seguridad" de la imagen de RAM original que el PC envía al entrenador. En este caso siempre se tiene la seguridad de que los datos están tras el reset como se declararon en el programa ensamblador.

La elección tomada ha sido la segunda opción, ya que a cambio de un retardo de unos segundos cada vez que se reinicia la máquina virtual Algorítmez del entrenador y un pequeño incremento en complejidad de hardware (En el punto 4.3.3.4. se ve más detenidamente) se libra al programador de cualquier preocupación.

Por lo tanto, para un programador de Algorítmez, las 64Kbytes de memoria son de tipo RAM y puede suponer que tras el reset ya contiene el programa ensamblado completo.

4.3.2.2. Puertos

Se han añadido nuevos puertos accesibles desde programas Algorítmez. El listado completo de puertos y su correspondencia con los periféricos es el siguiente:

Puerto	Periférico	Uso
0	Teclado	P. de estado. Bit 0: Preparado. Bit 1: Permitir interrupciones
1		Entrada: Carácter de tecla pulsada.
2	Pantalla	P. de estado. Bit 0: Preparado. Bit 1: Permitir interrupciones
3		Salida: Carácter a imprimir.
4	UART	P. de estado. Bit 0: Byte recibido. Bit 1: Permitir interrupciones (Ver abajo más explicaciones)
5		Entrada y salida: Byte recibido o byte a enviar, respectivamente.
6	Switchs	P. de estado. Bit 0: Preparado. Bit 1: Permitir interrupciones
7		Entrada: Estado de los 8 switchs.
8	LEDs	No usado
9		Salida: Nuevo estado de los 8 LEDs.

TABLA 12: Puertos Algorítmez en el entrenador.

Cualquier salida a un puerto no definido será ignorada, así como las lecturas de puertos no definidos o de sólo escritura, se leerán como cero.

Se ha guardado compatibilidad con los dos periféricos de los emuladores software, el teclado y la pantalla. La UART emulada por software permite la comunicación de programas en Algorítmez con cualquier otro equipo a través del puerto COM2 del entrenador. Los LEDs y los switchs son dispositivos instalados en el entrenador de forma permanente y pretenden servir de puertos de salida y entrada, respectivamente, fácilmente visibles y manipulables manualmente por el usuario para la depuración de aplicaciones.
A continuación se ve en más detalle el uso de cada puerto, el significado de cada bit y si el puerto es de entrada y salida (E/S), solo entrada (E) o solo salida (S).

Puerto[0]: Teclado – Estado (E/S)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	X						INT	PR
Valor inicial	0						0	0

Solo se puede escribir el bit INT, ignorando la escritura en los demás bits.

- PR: Bit de preparado. Cuando esta a 1, significa que se ha pulsado una tecla y esta lista para ser leída.
- INT: Máscara de interrupciones. Si esta a 0 inhibe la petición de interrupción de éste dispositivo cuando su estado sea de preparado.

Puerto[1]: Teclado – Datos (E)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	CARÁCTER							
Valor inicial	0x00							

Al leer este puerto se obtiene el código ASCII del carácter que se ha pulsado en el teclado. Ver las limitaciones respecto a los caracteres leídos desde el teclado en el punto 4.3.5.4.
Al realizar una operación de lectura de éste puerto, se pone a cero automáticamente el bit PR del puerto de estado del teclado.

Puerto[2]: Pantalla – Estado (E/S)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	X						INT	PR
Valor inicial	0						0	1

Solo se puede escribir el bit INT, ignorando la escritura en los demás bits.

- PR: Bit de preparado. Cuando esta a 1, significa que el periférico esta preparado para enviar un nuevo carácter a la pantalla.
- INT: Máscara de interrupciones. Si esta a 0 inhibe la petición de interrupción de éste dispositivo cuando su estado sea de preparado.

Puerto[3]: Pantalla – Datos (S)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	CARÁCTER							
Valor inicial	0x00							

Al escribir en este puerto, se envía el byte como carácter a imprimir a la pantalla virtual de Algorítmez. Tras hacer esto se pone a cero el bit de preparado del puerto de estado de la pantalla, que se volverá a poner a uno tras un período de tiempo, que se ha elegido de 50 instrucciones ejecutadas.
Si se envía un carácter 0x00 a la pantalla, esta se limpiará y se colocará el cursor en la posición inicial. Si se envía el carácter 0x01 se realizará un scroll de una línea hacia arriba. Si se envía un carácter normal y el cursor esta al final de una línea, se saltará automáticamente al comienzo de la siguiente, así como se realizará scroll de pantalla si es necesario.

Puerto[4]: UART – Estado (E/S)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	X	VEL					INT	PR
Valor inicial	0	0x00					0	1

Se puede escribir en los bits 1 a 6, ignorando la escritura en los demás bits.

- PR: Bit de preparado. Cuando esta a 1, significa que se ha recibido un carácter por el puerto serie COM2, y este está almacenado en el buffer accesible leyendo el puerto de datos.
- INT: Máscara de interrupciones. Si esta a 0 inhibe la petición de interrupción de éste dispositivo cuando su estado sea de preparado.
- VEL: Velocidad de bit. La relación entre la velocidad de bit que usa la UART para recibir y enviar, y el campo VEL se ha programado de la siguiente forma:

$$VEL = \frac{7376800}{BAUD \cdot 64} - 2 \qquad BAUD = \frac{7376800}{(VEL + 2) \cdot 64}$$

Donde 7376800 = 7.3768 MHz es la frecuencia del cristal con el que funciona el microcontrolador, escogida especialmente para ser múltiplo de todas las frecuencias estándar de transmisión serie y BAUD es la velocidad de bit en bits por segundo.
La siguiente tabla contiene ya calculados los valores de VEL para valores estándar de BAUD:

Velocidad (bps)	**Valor de "VEL"**
57600	0
38600	1
19200	4
9600	10
4800	12

TABLA 13: Valores de VEL para velocidades estándar.

Puerto[5]: UART – Datos (E/S)

Bit	7	6	5	4	3	2	1	0
Significado	BYTE							
Valor inicial	0x00							

Si se lee este puerto, se obtiene el último byte recibido por el puerto serie COM2 del entrenador. Sólo se debería leer este puerto si se ha comprobado antes que ha llegado un byte nuevo, lo que se verá en el bit preparado del puerto de estado que deberá estar a uno. Al realizar la lectura sobre este puerto, se borra automáticamente el bit de preparado del puerto de estado.

Si se escribe en este puerto, se envía inmediatamente el byte por el puerto serie COM2 a la velocidad programada en el puerto de estado. Debido a las limitaciones de la UART emulada por software, se para la ejecución del programa Algorítmez mientras se está enviando este byte, lo que no debería ser un problema ya que por ejemplo a 57600bps eso representa una parada de solo 134μs.

Puerto[6]: Switchs – Estado (E/S)

Bit	7	6	5	4	3	2	1	0
Significado	X						INT	PR
Valor inicial	0						0	0

Solo se puede escribir el bit INT, ignorando la escritura en los demás bits.

- PR: Bit de preparado. Cuando esta a 1, significa que se ha detectado un cambio en el estado de los switchs.
- INT: Máscara de interrupciones. Si esta a 0 inhibe la petición de interrupción de éste dispositivo cuando su estado sea de preparado.

Si el estado tras el reset de los switchs es distinto a 0x00, el bit de preparado se activará inmediatamente. Para que se dispare el bit de preparado de este puerto, el cambio en el

estado de los switchs debe durar más que el tiempo de ejecución de una instrucción Algorítmez. Ver el punto 4.3.8. para detalles sobre la velocidad del entrenador.

Puerto[7]: Switchs – Datos (E)

Bit	7	6	5	4	3	2	1	0
Significado	Estado de los 8 micro-switchs							
Valor inicial								

Una lectura en este puerto conlleva una lectura física inmediata del puerto C del microcontrolador, cableado en el entrenador a un micro switchs de 8 interruptores. Estos deben hacer contacto con masa al activarse, ya que se han activado internamente las resistencias de tirón a Vcc en cada uno de los pins del puerto, de forma que al no cerrarse un interruptor, ese bit se lee como un uno.
Al leer este puerto se pone a cero el bit de preparado del puerto de estado.

Puerto[8]: LEDs – Estado (E)

Bit	7	6	5	4	3	2	1	0
Significado	X							
Valor inicial	0							

Este puerto no se usa y siempre es leído como 0x00.

Puerto[9]: LEDs – Datos (S)

Bit	7	6	5	4	3	2	1	0
Significado	Valor a mostrar en los 8 leds							
Valor inicial	0x00							

Al escribir un byte en este puerto se saca este inmediatamente por el puerto A del microcontrolador, donde en el entrenador se han colocado 8 diodos LEDs, cableados de forma que se enciendan cada uno al poner un 1 en su bit correspondiente.

4.3.2.3. Tabla de periféricos

Al añadir nuevos periféricos a Algorítmez, la tabla de periféricos en memoria ha cambiado respecto a las especificaciones del simulador software, al reflejar estos nuevos periféricos. Con esto se permite que una rutina de consulta de interrupciones por vector común pueda detectar interrupciones en los cuatro periféricos capaces de generar una interrupción:

Fila	Dir E/S (1)	Estado (1)	Zona E/S (2)	Marco (1)	DF (1)	NBYP (2)	Zona user(2)	Dir aviso(2)	NBL (2)	NBYT (2)	PUNTES (2)	PUNTZU (2)
0	2	0x80	0	0	0	0	0	0	0	0	0	0
1	0	0x80	0	0	0	0	0	0	0	0	0	0
2	4	0x80	0	0	0	0	0	0	0	0	0	0
3	6	0x80	0	0	0	0	0	0	0	0	0	0
4	0xFF	0	0	0	0	0	0	0	0	0	0	0

TABLA 14: La tabla de periféricos en el entrenador de Algorítmez.

Las filas se corresponden con los periféricos:

- Monitor. Puerto de estado: 2.
- Teclado. Puerto de estado: 0.
- UART. Puerto de estado: 4.
- Switchs. Puerto de estado: 6.

4.3.2.4. Pantalla

Por limitaciones del hardware, la pantalla virtual de Algorítmez es una zona de memoria RAM donde se guardan las salidas por el puerto de la pantalla (que se puede visualizar en el LCD o no en cada momento) y contiene solamente 2 filas de 16 caracteres cada una, al igual que el display usado en el proyecto.

4.3.3. Hardware

La principal prioridad usada al decidir el hardware ha sido la sencillez. Siguiendo esta filosofía, se ha conseguido un sistema final con gran capacidad de procesamiento y almacenamiento comparado con el mínimo y económico hardware usado.

Como es lógico, la sencillez tiene un lado negativo. En este caso ha sido la velocidad de simulación, reinicio y descarga de programas, que sin llegar a ser demasiados lentos, podrían haberse mejorado notablemente usando sistemas más complejos y caros.

En esta sección se analizan cada uno de los subsistemas hardware y las decisiones tomadas en sus implementaciones y las elecciones entre las distintas alternativas disponibles en el mercado, llegando finalmente a un esquema completo del sistema.

4.3.3.1. Microcontrolador

Se ha decidido usar un microcontrolador de 8 bits como el AT90S8515 del fabricante Atmel, como base de todo el diseño. Las ventajas de este microcontrolador son:

- Fácil disponibilidad y muy buena relación precio / potencia.
- Reprogramable cientos de veces, lo que ha sido imprescindible durante la creación y depuración del prototipo.
- Alta eficacia, gracias a la arquitectura Hardvard con procesamiento paralelo de instrucciones y encauzamiento de 2 etapas, permite alcanzar una relación MIPS/Mhz muy cercana a la unidad.
- Una UART y gran cantidad de puertos de entrada y salida. Esto es importante, ya que ha permitido eliminar cualquier otro chip de interfaz auxiliar para el manejo de periféricos.
- Permite dos entradas de interrupciones externas, ambas usadas en el diseño.
- Capacidad de memoria suficiente para la aplicación. Aunque este microcontrolador esta preparado para el uso de memorias RAM estáticas externas, se ha evitado para poder disponer de un mayor número de puertos de E/S libres. Las memorias de que dispone internamente este microcontrolador son:
  - 8 Kbytes de memoria flash de programa. Como las instrucciones son de 16 bits, existe espacio para un máximo de 4096 instrucciones, aunque hay que descontar el espacio de tablas y constantes.
  - 512 bytes de RAM interna.
  - 512 bytes de EEPROM interna.
- El fabricante proporciona un entorno IDE (Integrated Development Environment) con ensamblador y simulador, de forma gratuita a través de su web, que se ha usado para el ensamblado y depuración del firmware.

Para proporcionar el espacio de memoria RAM de Algorítmez, se hace necesario el uso de memorias externas, como se detalla en el punto 4.3.3.4.

4.3.3.2. Display LCD

La mayoría de los displays LCDs de texto actuales, son compatibles con el estándar *de facto* HD44780 de Hitachi. El interfaz de estos LCDs esta formado por 14 señales, como se ve en la siguiente figura:

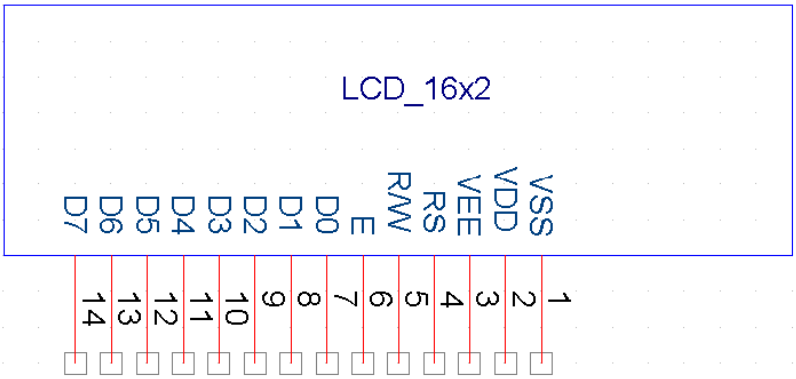


FIGURA 15. Interfaz estándar del LCD

- VDD, VSS: Alimentación y masa respectivamente.
- VEE: Entrada de ajuste del contraste.
- D0-D7. Es un bus de 8 bits bidireccional.
- RS (Register selection) : Selección de registro.
- R/W (Read/Write): Lectura (1) / escritura (0).
- E (Enable): Indicador de datos validos en bus. Activo a nivel alto.

Aunque existen LCDs de diversos tamaños (1, 2 o 4 líneas, 8, 16, 24, 32 o 40 caracteres por línea) todos aceptan el mismo protocolo y conjunto de comandos. Además, todos suelen soportar los modos de comunicación a bus completo y a medio bus (4 bits).
En el caso del entrenador, se usa un LCD de 16 caracteres y 2 líneas, y el modo de bus a 4 bits, por requerir un menor número de puertos de E/S en el microcontrolador que pueden usarse para otros periféricos. La forma de enviar datos por el bus de 4 bits, es colocando en el bus el primer nibble a enviar, dar un pulso a E y a continuación repetirlo con el nibble bajo. A continuación se detallan en forma de tabla todas las

operaciones que se pueden realizar sobre el display y que señales hay que activar para ejecutarlas:

OPERACIÓN	**SEÑALES**									
	RS	**R/W**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Borrar display.	0	0	0	0	0	0	0	0	0	1
Cursor a posición inicial.	0	0	0	0	0	0	0	0	1	X
I/D:Especifica dirección de entrada de texto: 0=IZQ, 1=DER. S: Realizar scroll automático de display	0	0	0	0	0	0	0	1	I/D	S
D: Display activado (1) o desactivado (0) C: Mostrar (1) o no (0) el cursor B: Cursor parpadeando (1) o fijo (0)	0	0	0	0	0	0	1	D	C	B
S/C: Scroll(1) o mover cursor(0) al introducir texto R/L: Dirección scroll 1=DER, 0=IZQ	0	0	0	0	0	1	S/C	R/L	X	X
DL: 1= Modo 8 bits, 0= Modo 4 bits N: 0= Una línea, 1= Dos líneas	0	0	0	0	1	DL	N	F	X	X
Indica dirección a escribir en la CGRAM	0	0	0	1	A	A	A	A	A	A
Indica dirección a escribir en la DDRAM	0	0	1	A	A	A	A	A	A	A
Leer estado de display: BF=1 ocupado, BF=0, libre. Devuelve también la dirección actual en CG/DDRAM	0	1	BF	A	A	A	A	A	A	A
Escribe dato	1	0	D	D	D	D	D	D	D	D
Leer dato	1	1	D	D	D	D	D	D	D	D

TABLA 15: Operaciones soportadas por el LCD

Se dice que estos LCDs son “inteligentes”, porque disponen en su interior de un controlador que procesa todos estos comandos, además de realizar ciertas tareas automáticamente, como incrementar la posición del cursor al escribir un carácter, parpadeo automático del cursor, etc... Es éste controlador el que realmente controla la gran cantidad de líneas necesarias para mostrar en el display cada uno de los píxels.

Los displays contienen dos zonas de memoria:

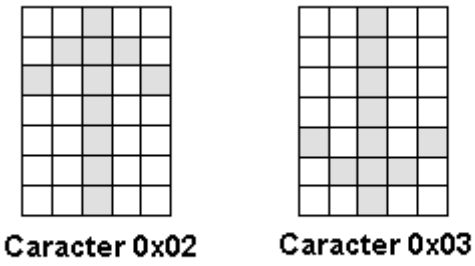
- DDRAM: Display Data RAM. Guarda el carácter almacenado para cada posición del display.
- CGRAM: Carácter Generator RAM. Usados para almacenar hasta 8 caracteres definidos por el usuario. La resolución es de 5x7 pixels por carácter.

Hay que decir que el código de caracteres de los LCDs es el ASCII de 7 bits, mientras que los 128 caracteres superiores son en su mayoría caracteres japoneses, por lo que algunos caracteres especiales del español no podrán mostrarse con sus códigos ASCII extendidos:

FIGURA 16. Conjunto de caracteres del LCD

Los códigos 0x00 a 0x07 son los correspondientes a caracteres definidos por el usuario en la CGRAM. Los códigos 0x08 a 0x0F son equivalentes a los códigos 0x00 a 0x07.

En el entrenador se han definido dos de estos caracteres que resultan útiles para los menús:



En el siguiente cronograma se muestran casos del uso de bus a 4 bits, como es el caso del entrenador:

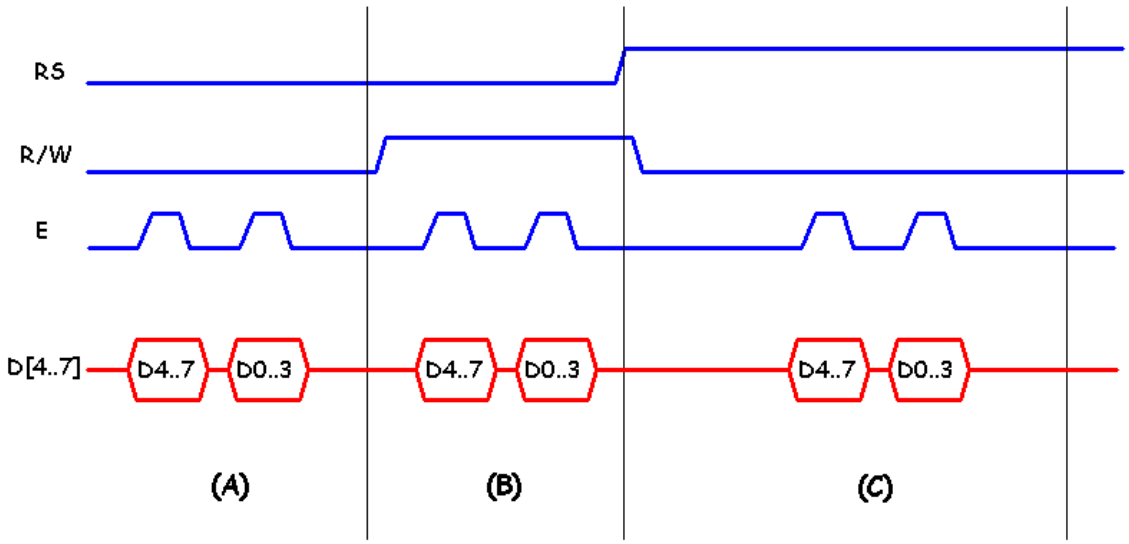


FIGURA 17: Cronogramas de bus del LCD

En el caso A, se esta mandando una instrucción al display, y se ve como hay que separarla en dos *nibbles*. Así es como se envían en el entrenador todas las instrucciones al display (borrar LCD, posicionar cursor, etc...).

El caso B, es la lectura del estado del *display*. Como se veía en la tabla de comandos, el bit 7 indica si el LCD ha terminado de procesar la última operación. Este tiempo de proceso puede ir desde unos 40μs, hasta algunos milisegundos, dependiendo de la operación y del modelo del display. Por eso, en el entrenador siempre se consulta este bit tras realizar una operación, y no se permite seguir hasta que el LCD haya terminado, ya que si se intenta enviar otro comando o dato mientras está ocupado, este se perderá.

El caso C, muestra como se envían "datos", es decir, los caracteres a almacenar en la posición actual del cursor del display. Se pueden mandar varios caracteres consecutivamente, y se almacenarán en posiciones consecutivas de la pantalla.

Hay que indicar además, que las direcciones de memoria DDRAM correspondientes al espacio visible del display de 16x2 caracteres son las siguientes:

Línea 1: Posiciones 0x00 a 0x0F
Línea 2: Posiciones 0x40 a 0x4F

Esto quiere decir que cuando se llena una línea, si se siguen enviando caracteres estos no aparecerán en la siguiente línea, sino que esto se tiene que realizar por software.

4.3.3.3. Teclado PS/2

Debido al amplio uso de teclados estándar para PC y a su bajo coste comparado con otros tipos de teclados mucho menos versátiles, menos económicos y con interfaces más complicadas, se ha decidido usar un teclado estándar tipo PS/2.

Estos teclados incorporan cierta "inteligencia", lo que los hace relativamente fáciles de usar. Incorporan un controlador conectado a una línea serie síncrona, que monitoriza el estado de las teclas.

Cada tecla tiene asignada un *scan code*, como se ve en la figura:

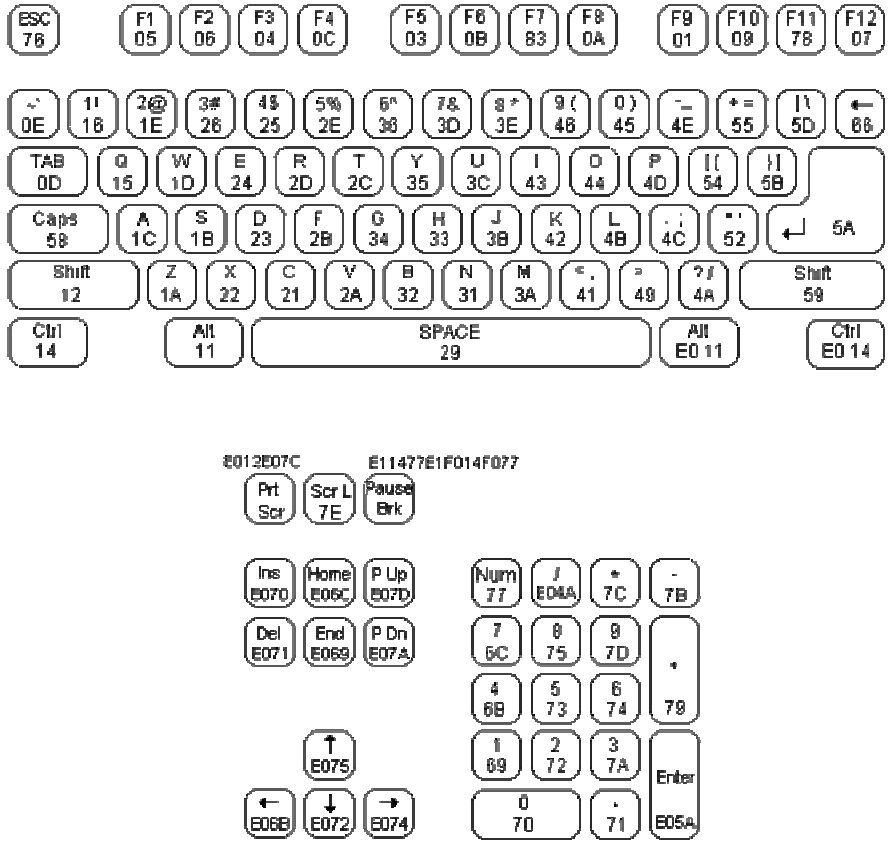


FIGURA 18. Mapa de *scan codes* del teclado

La mayoría de las teclas más comunes (caracteres, dígitos, teclas de función F1..F10,etc...) tienen *scan codes* de 1 byte, mientras que otras teclas tienen códigos de varios bytes. La mayoría de estas teclas extendidas, usan el “prefijo” 0xE0.

Cuando se pulsa una tecla, el teclado comienza a transmitir a intervalos regulares (velocidad de repetición de tecla) el *scan code* de esa tecla, y cuando se suelta, envía un byte 0xF0 seguido del *scan code* de la tecla que se ha soltado, para indicar que esa tecla ya no está pulsada.

Las señales usadas para la comunicación entre el sistema y el teclado son:

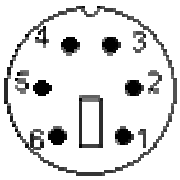


FIGURA 19. Conexiones PS/2

1. **Clk -** Señal de reloj
2. **Gnd** - Masa
3. **Data** - Línea de datos serie
4. No usado.
5. **Vcc -** Alimentación a +5V
6. No usado.

Como se ha dicho arriba, esta comunicación es serie síncrona, usando la línea *Data* para enviar los bits, y la línea *Clk* para el reloj. Ambas líneas son bidireccionales, lo que se consigue usando una resistencia de tirón a VCC en cada línea, y usando salidas de drenador abierto en ambos extremos.

En el siguiente cronograma se muestra como el teclado envía un byte:

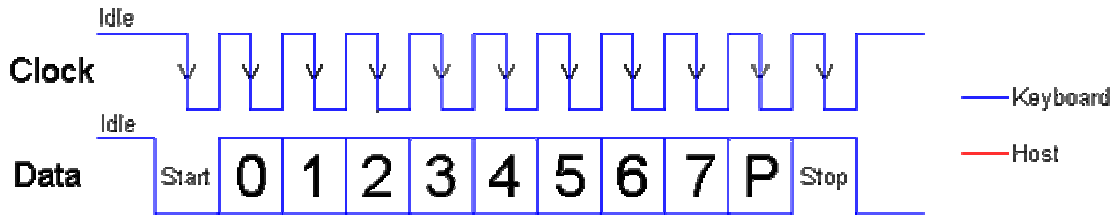


FIGURA 20. Cronograma de envío de un byte por el teclado

Cuando el teclado quiere enviar un byte, primero pone *Data* a nivel bajo para generar un bit de comienzo, y genera una señal de reloj para que el otro extremo (el entrenador en este caso) pueda muestrear fácilmente los bits en el flanco de bajada. La frecuencia de este reloj suele ser baja, en torno a 20 o 30 KHz. Se usa un bit de paridad impar.

En el entrenador, se ha colocado la línea *Data* del teclado a una entrada de interrupción externa del microcontrolador (La línea INT0), con lo que se detecta cuando comienza a recibir un byte sin necesidad de realizar espera activa sobre esta línea. Las interrupciones llevan a la ejecución de una rutina que se encarga de procesar los avisos de tecla pulsada y tecla soltada, y transformando los scan codes a ASCII. Esto se verá más detenidamente en el punto 4.3.5.5.

4.3.3.4. Bancos de memoria externa

Por los motivos que se explican en 4.3.2, se han usado dos bancos de memoria EEPROM serie de 64Kb, accesible por bus I2C. La memoria EEPROM (Electrical Erasable Programable ROM) se caracteriza por tiempos cortos de lectura, y tiempos de escritura de 5 ms aproximadamente, aunque existe un modo de escritura de página que en el mismo tiempo es capaz de escribir bloques de hasta 32 o 128 bytes (según el modelo de la memoria). Sin embargo, se ha elegido frente a soluciones basadas en RAM estática por su mayor sencillez y por su característica de retención de datos sin alimentación. A continuación se detallan el interfaz y el esquema interno que ofrece esta memoria:

FIGURA 21. Memoria 24C512. Patillaje

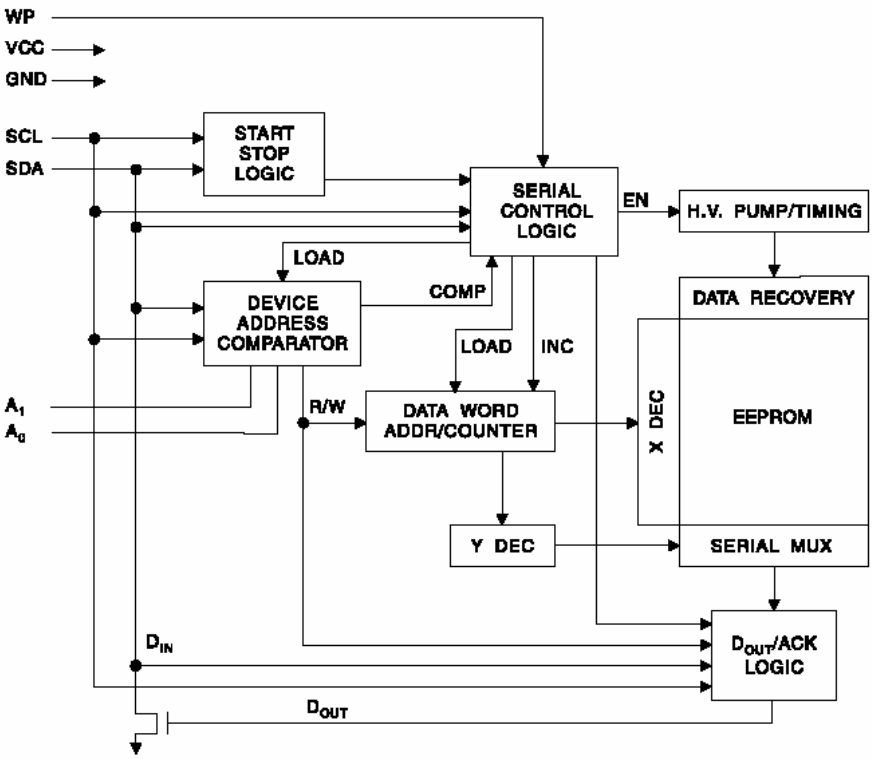
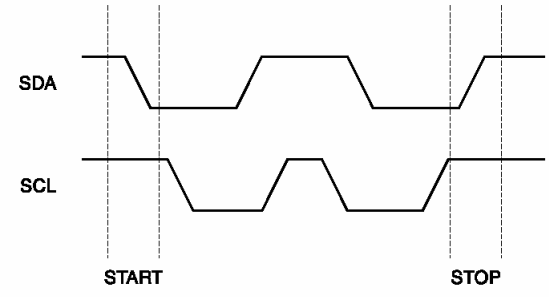


FIGURA 22. Memoria 24C512. Esquema interno

- VCC y GND: Alimentación a 5V y masa, respectivamente.
- WP (Write Protect). Protección de escritura si está a nivel lógico alto. En el diseño esta cableado a masa, ya que es necesario escribir y leer.
- SCL y SDA. Las líneas de reloj y datos, respectivamente, del bus I2C.
- A0,A1: Forman una dirección de 2 bits para el chip. Estas direcciones permiten usar hasta 4 dispositivos I2C conectados a un mismo bus. En el entrenador se han usado dos memorias, una con la dirección b'00 y otro con la dirección b'01. Al realizar una operación sobre el bus, solo se habilita el chip cuya dirección I2C coincida con la de la petición.

El protocolo I2C es un protocolo *Master-Slave* (maestro-esclavo), donde se define diferenciadamente el papel de "maestro" (en este caso el microcontrolador), el cual interroga a un "esclavo" (las memorias). Para el esclavo, ambas líneas SDA y SCL son entradas (inicialmente), mientras que para el maestro deben ser de salida, aunque la línea SDA es de drenador abierto en ambos extremos, de forma que se necesita una resistencia de tirón a VCC para conseguir el nivel lógico alto cuando ningún extremo esté enviando un nivel bajo. En el entrenador, se usan las resistencias de tirón programables internas al microcontrolador, evitando componentes discretos externos.

Para iniciar la comunicación, el maestro debe ejecutar una secuencia de "inicio" en el bus, así como para finalizar debe provocar una secuencia de "final". Estas transiciones se definen como transiciones de la señal SDA mientras SCL esta en estado alto, mientras que normalmente SDA solo puede cambiar durante el período bajo de SCL:



Tras enviar esta señal de inicio, el maestro envía un byte que indica el tipo de operación a realizar (lectura o escritura) y la dirección I2C del dispositivo al que se dirige. Todos los bytes se envían con el MSB primero. Este byte se forma así con los siguientes bits:

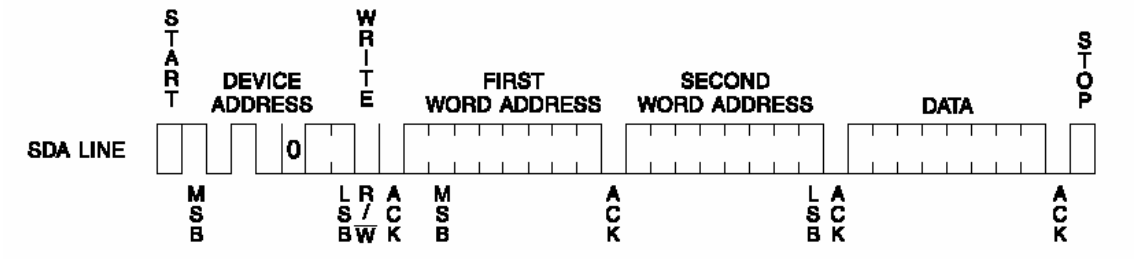
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	0	1	0	0	A1	A0	R / W

Donde A1:A0 son los bits de dirección I2C y R/W es 1 en caso de lectura y un 0 en caso de escritura.
A cada byte que se envíe, en sentido maestro – esclavo o viceversa, le sigue un bit de asentimiento, que el otro extremo deberá dar para asegurar que la comunicación se esta realizando correctamente. Durante este período de bit, la línea SDA se deja a nivel alto, debiendo el otro extremo ponerla a nivel bajo para dar un asentimiento positivo de recibo. De esta forma las operaciones se realizan mediante secuencias: Envío de byte, ciclo de asentimiento.

Los comandos que se han usado en este proyecto para la memoria AT24C512 han sido los siguientes:

Escritura aleatoria

Permite la escritura de un solo byte en una dirección específica de la memoria. Se realiza con la siguiente secuencia:



Que se puede separar en las siguientes etapas:

1. El microcontrolador genera una señal de inicio.
2. Se envía a la memoria el byte de comando, indicando su dirección I2C y el bit R/W a cero indicando escritura.
3. Se envían el byte de dirección alto y después el bajo (Direcciones de 16 bits para espacio de direcciones de 64 Kb)
4. Se envía el byte que se quiere guardar en esa dirección.
5. Se genera una señal de stop, iniciando la grabación en la EEPROM.

La temporización de la escritura es importante, al no poder realizar ninguna otra operación sobre la memoria mientras que esta es llevada a cabo. Se pueden usar dos técnicas:

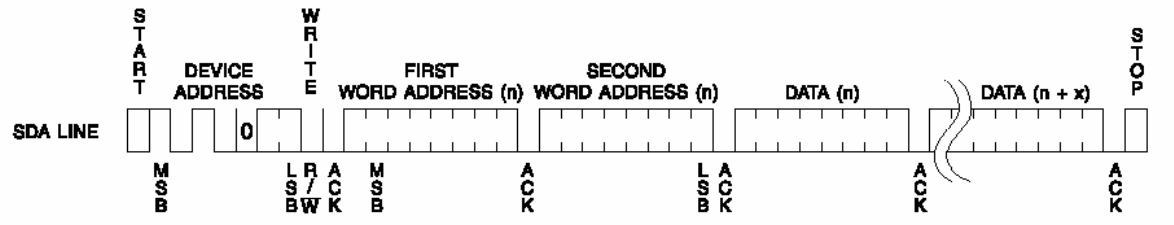
- Espera pasiva. Se espera un mínimo de 5 ms tras enviar una petición de escritura, asegurando que se ha llevado a cabo completamente antes de enviar otra petición.
- Espera activa. Se aprovecha que la memoria no devuelve el asentimiento positivo mientras está realizando una grabación, para determinar por *polling* cuando ha terminado.

Se ha implementado la segunda opción, ya que permite aprovechar al máximo los recursos.

Escritura de página

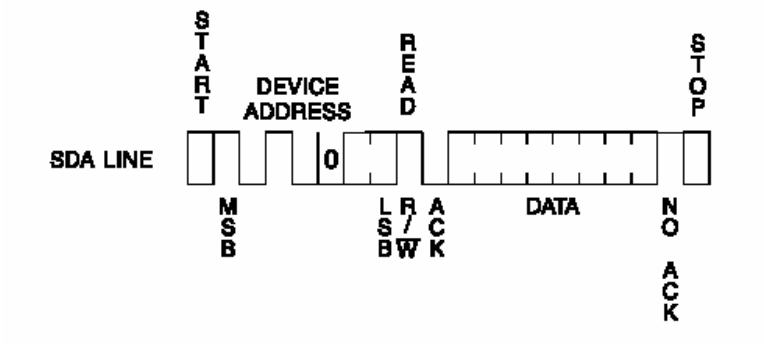
Permite la escritura simultánea de hasta 128 bytes en direcciones consecutivas de la memoria. Primero se envían los bytes y son guardados en un buffer temporal. Tras enviar la secuencia de stop, se inicia la secuencia de grabación de forma paralela de tantos bytes como se hayan enviado.

Representado en cronograma:



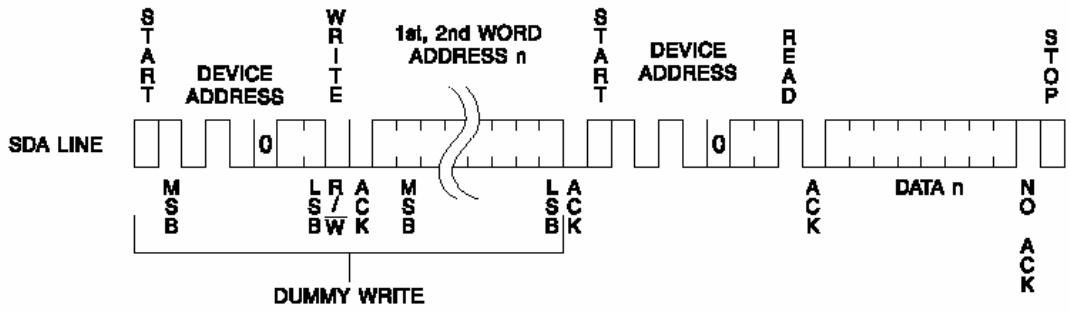
Lectura de posición actual

En las operaciones de escritura, se envía una dirección a la memoria. Esta dirección es guardada en un registro interno al chip. Con el siguiente comando, se puede leer la dirección apuntada por este registro, incrementándola en una unidad posteriormente:



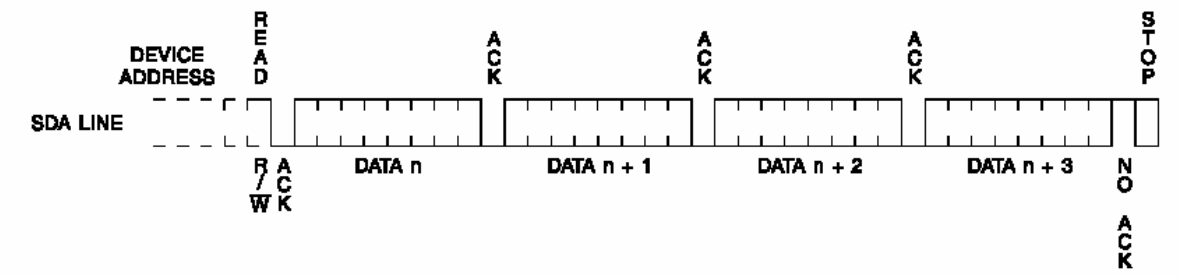
Lectura de posición aleatoria

En realidad se compone de una secuencia de escritura, abortada tras el envío de la dirección, seguida de una secuencia de bit de inicio, y una lectura de posición actual. Con el comando de escritura abortado se consigue guardar en el registro de direcciones la dirección pedida:



Lectura secuencial

Permite la lectura de cuantos bytes consecutivos se deseen. Esta modalidad implica una gran mejora en el tiempo medio de acceso a memoria, ya que se evita el envío de un byte de comando de lectura por cada byte leído:



Cuando el microcontrolador lea el último byte, debe no dar su asentimiento para que la memoria sepa que ha terminado la operación.

En el entrenador se han usado dos de estos chips AT24C512, de 64Kbs cada uno. Se han conectado al mismo bus I2C, usando para direccionarlos las direcciones 0 y 1:

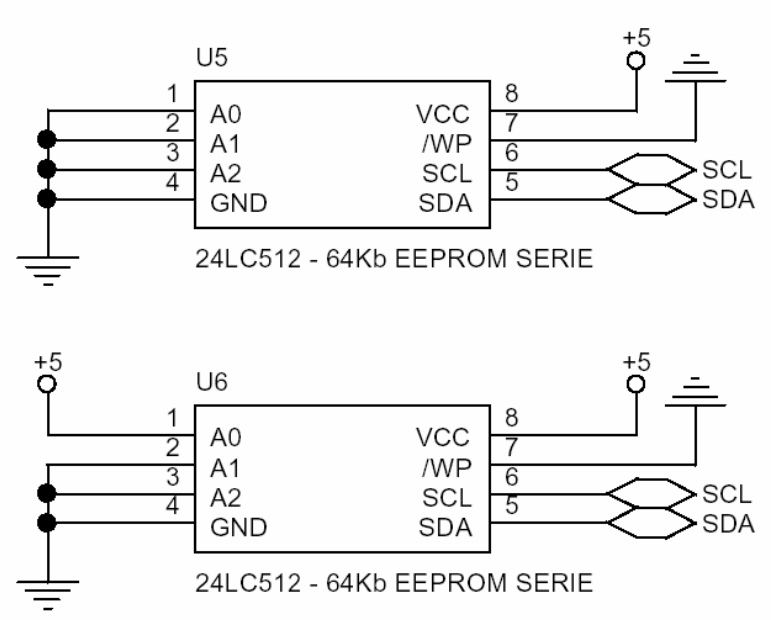


FIGURA 23: Conexiones de las memorias externas.

4.3.3.5. Switchs y LEDs

El entrenador dispone de dos puertos accesibles para los programas Algorítmez que se corresponden con puertos hardware del microcontrolador de la siguiente forma:

Puerto Algorítmez	**Puerto del microcontrolador**
7 (Entrada)	PORT.C
9 (Salida)	PORT.A

Es decir, estos puertos son de propósito general, uno de entrada y otro de salida, pudiendo ser usados para controlar cualquier hardware externo, aunque en el sistema entrenador se ha conectado el puerto de entrada a 8 interruptores en un microswitch, y el puerto de salida a 8 diodos LEDs, en serie cada uno con una resistencia de 1KΩ, y hacia masa (se encienden con un nivel lógico alto). Estos elementos se han colocado en el frontal del entrenador para su fácil acceso y visualización, convirtiéndolos en elementos muy útiles para visualizar o introducir de forma fácil un valor de 8 bits desde un programa de Algorítmez.

Para el puerto de entrada se han habilitado las resistencias de tirón a Vcc internas, lo que quiere decir que siempre leerán un estado lógico alto por defecto, a menos que un dispositivo externo absorba corriente para generar un estado lógico bajo. Por ello los microswitchs se han conectado a masa, de forma que al cerrar uno de ellos, el valor del bit correspondiente del puerto 7 se leerá como un cero. Si el microswitch permanece abierto, el bit se leerá como uno debido a las resistencias de tirón.

4.3.3.6. Oscilador a cristal

El oscilador a cristal es la fuente de la señal de reloj para el microcontrolador. La frecuencia de este oscilador se ha elegido teniendo en cuenta estas ideas:

- Cuanto más alto sea, mayor será la velocidad del entrenador, lo que es deseable.
- El microcontrolador soporta hasta un máximo de 8Mhz recomendados por el fabricante.
- La frecuencia debe tener un valor que facilite las velocidades estándar de las comunicaciones por puerto serie.

Por todo esto, se ha elegido el valor 7.3728 Mhz que cumple con todos estos requisitos. El oscilador que se ha usado es uno basado en una puerta lógica inversora:

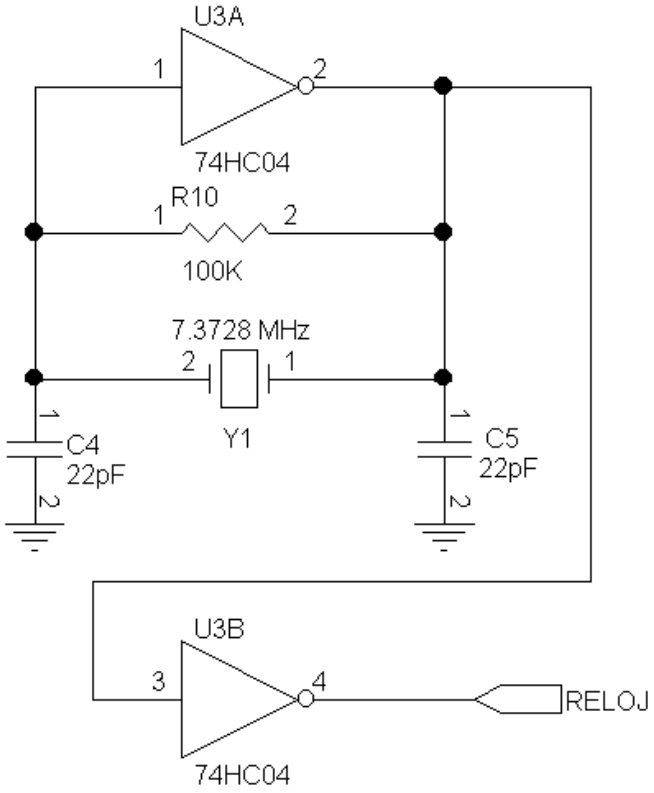


FIGURA 24. Circuito del oscilador a cristal

Donde el cristal de la frecuencia elegida se usa en modo paralelo. La resistencia entre la entrada y la salida ayuda a arrancar el oscilar al proveer realimentación, así como los condensadores ayudan a conformar el circuito resonante.

La señal se pasa por otra puerta inversora U3B que hace de buffer, y la señal resultante ya si puede distribuirse a lo largo de pistas del circuito relativamente largas sin problemas, mientras que las pistas que van a la entrada y salida de la puerta U3A se han mantenido lo más cortas posibles.

4.3.3.7. Puertos serie

El entrenador dispone de dos conectores de puerto serie, que se han llamado COM1 y COM2. Ambos conectores cumplen la norma RS-232 en cuanto a especificaciones física de conectores y niveles de señales, por lo que son compatibles con los puertos serie de cualquier PC o equipo de comunicaciones.
Los dos terminales son conectores del tipo DB-9 macho, y sólo se usan los siguientes pins:

- Pin 2 (Rd). Recepción de datos serie.
- Pin 3 (Td). Transmisión de datos serie.
- Pin 5 (Gnd). Masa.

Para la conversión de las señales entre los niveles lógicos RS232 y los TTL se ha usado el chip ST232 (compatible con el clásico MAX232), que a partir de una única fuente de alimentación a 5 voltios, genera internamente los ±10 voltios necesarios para las señales RS232. Para ello requiere de cuatro condensadores externos de 1μF, como se ve en el esquema que da el fabricante:

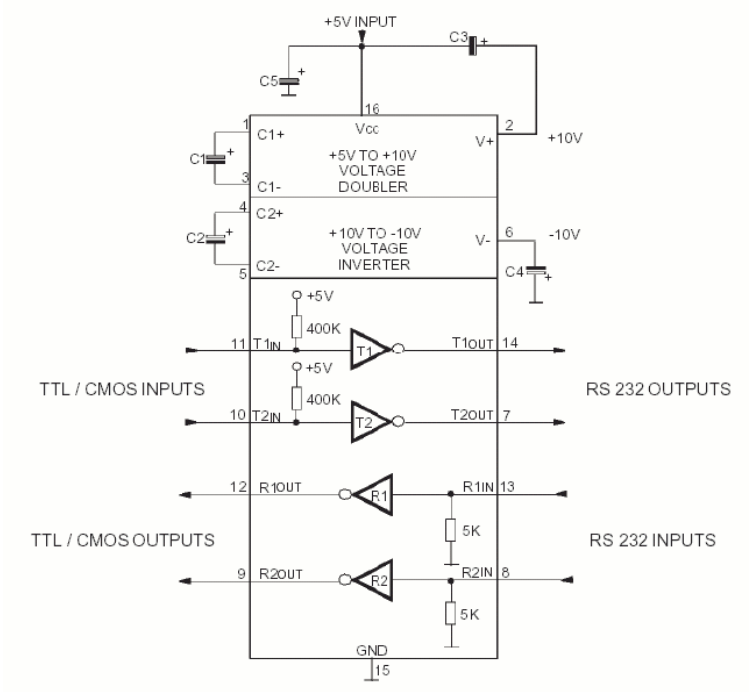


FIGURA 25. Esquema del chip ST232

De esta forma se usan dos de los conversores TTL → RS232 para las señales de transmisión de los puertos 1 y 2, y los otros dos conversores RS232 → TTL para las señales de recepción de ambos puertos, con lo que se usan los cuatro conversores que incorpora el chip.

Además se usan en el entrenador cuatro puertas lógicas NOT (las que no eran usadas por el oscilar a cristal) para proveer de unos indicadores por LEDs del estado de las líneas de transmisión y recepción de ambos puertos serie. Al ser una puerta NOT y estar los diodos LEDs hacia masa, estos se encenderán cuando la línea tenga un estado lógico bajo. Como normalmente el estado de la línea cuando está sin uso es de valor lógico alto, los LEDs permanecerán apagados, y sólo parpadearan cuando se transmitan bits a cero, lo que da una idea del tráfico de tramas que hay en cada momento, además de dar una confirmación de la correcta transmisión y recepción.

Por lo tanto, el esquema del ST232 junto con estas puertas y los diodos LEDs queda así:

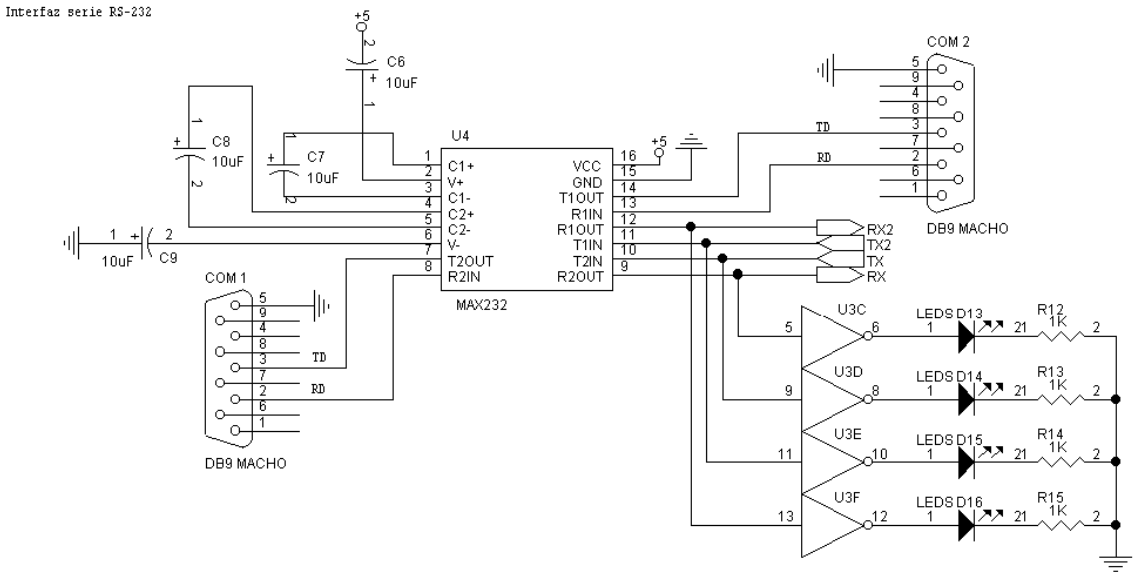


FIGURA 26. Esquema de la conexión del chip ST232

En cuanto a la conexión con el microcontrolador central, este dispone de una única UART, que se ha conectado al puerto COM1. Los puertos de los que hace uso la UART

son el PD0 (pin 10) para la recepción (RX en el esquema) y el PD1 (pin 11) para la transmisión (TX en el esquema).

Para el puerto serie COM2, se han usado puertos de propósito general, y la recepción y transmisión se realizan por software. En este caso el PD2 (pin 12) se usa para recepción (RX2 en el esquema) y el PD5 (pin 15) se usa para transmisión (TX2 en el esquema).
Se ha elegido usar el PD2 para recepción porque además es una entrada de interrupción (INT1), que se dispara al detectar un nivel lógico bajo, es decir: Un bit de comienzo en un byte recibido.

4.3.3.8. Fuente de alimentación

Se ha introducido una fuente de alimentación al entrenador, de forma que se evite la dependencia de un generador externo. Esta es sencilla y usa el clásico esquema transformador – rectificador – filtro – regulador, como se ve en el esquema:

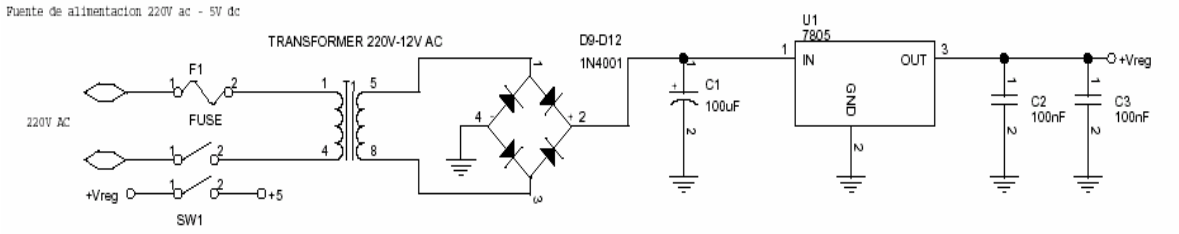


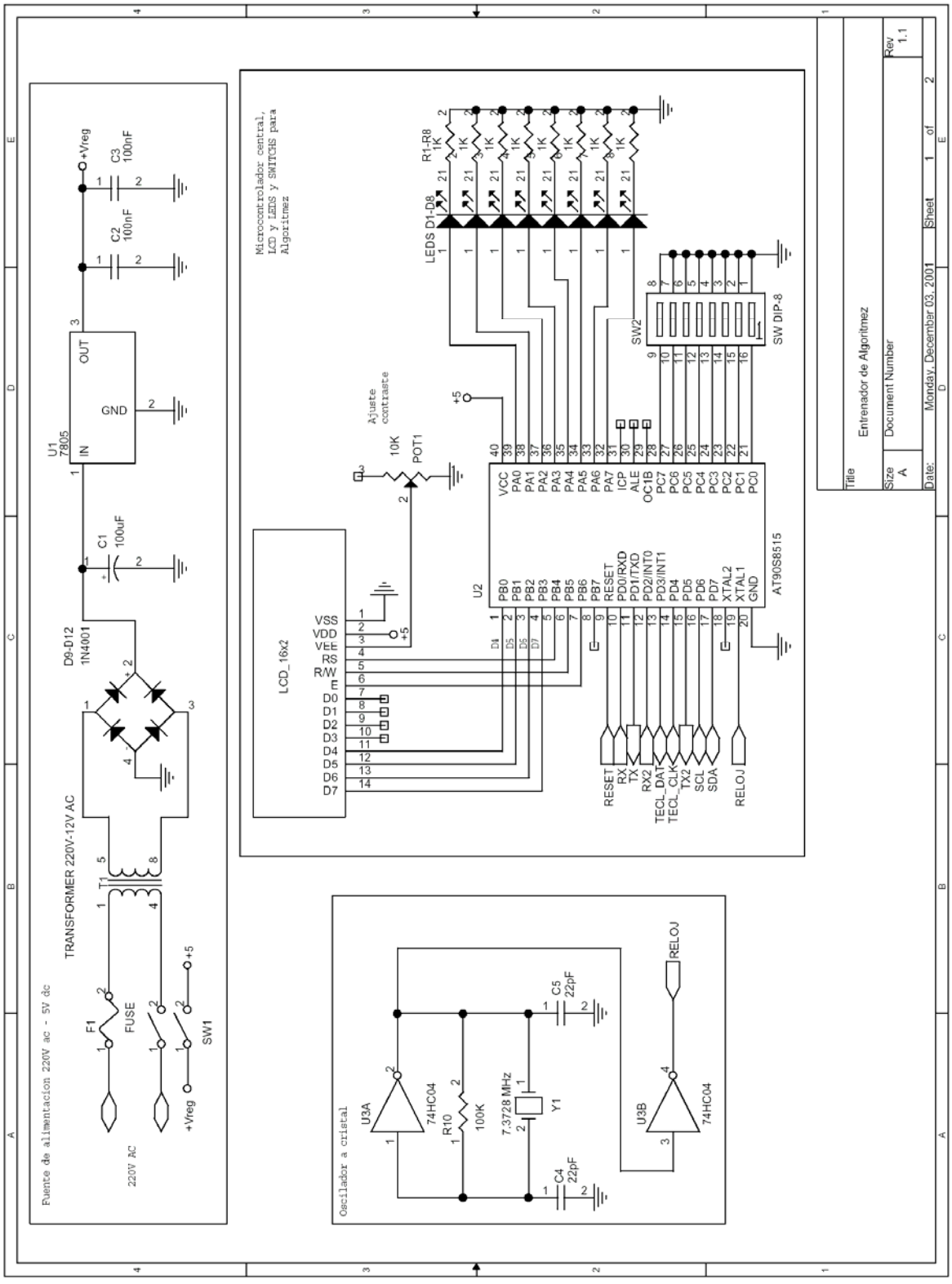
FIGURA 27. Esquema de la fuente de alimentación

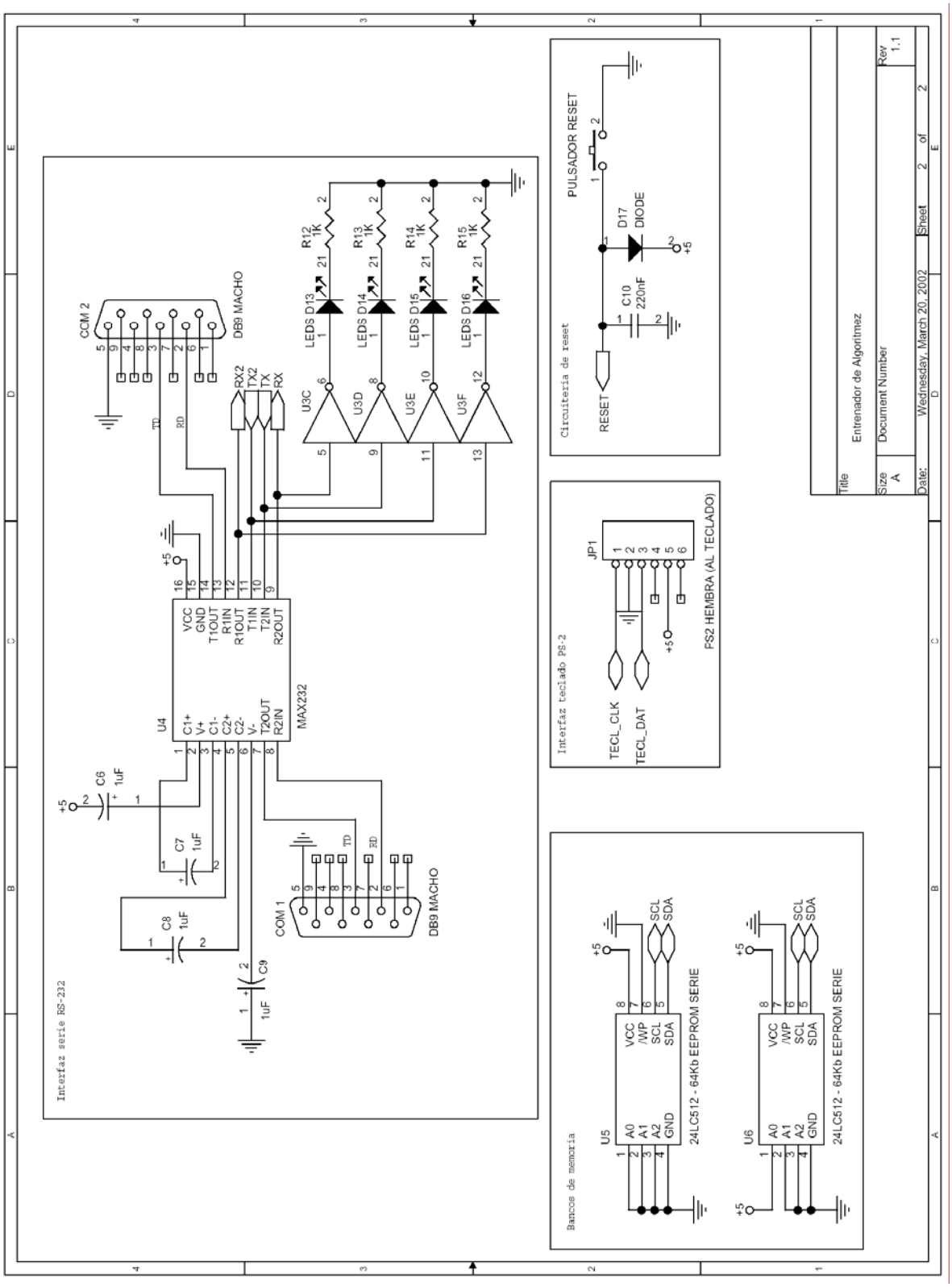
Usando como regulador un integrado 7805 (que cumple con las necesidades de potencia y regulación de voltaje) y posteriormente dos condensadores de filtro adicionales de 100nF cada uno. Estos se han colocado lo mas cerca posible del microcontrolador y del integrado del oscilador a cristal, evitando la emisión de transitorios a la línea de alimentación del sistema, causados por picos de corriente en los flancos del reloj.

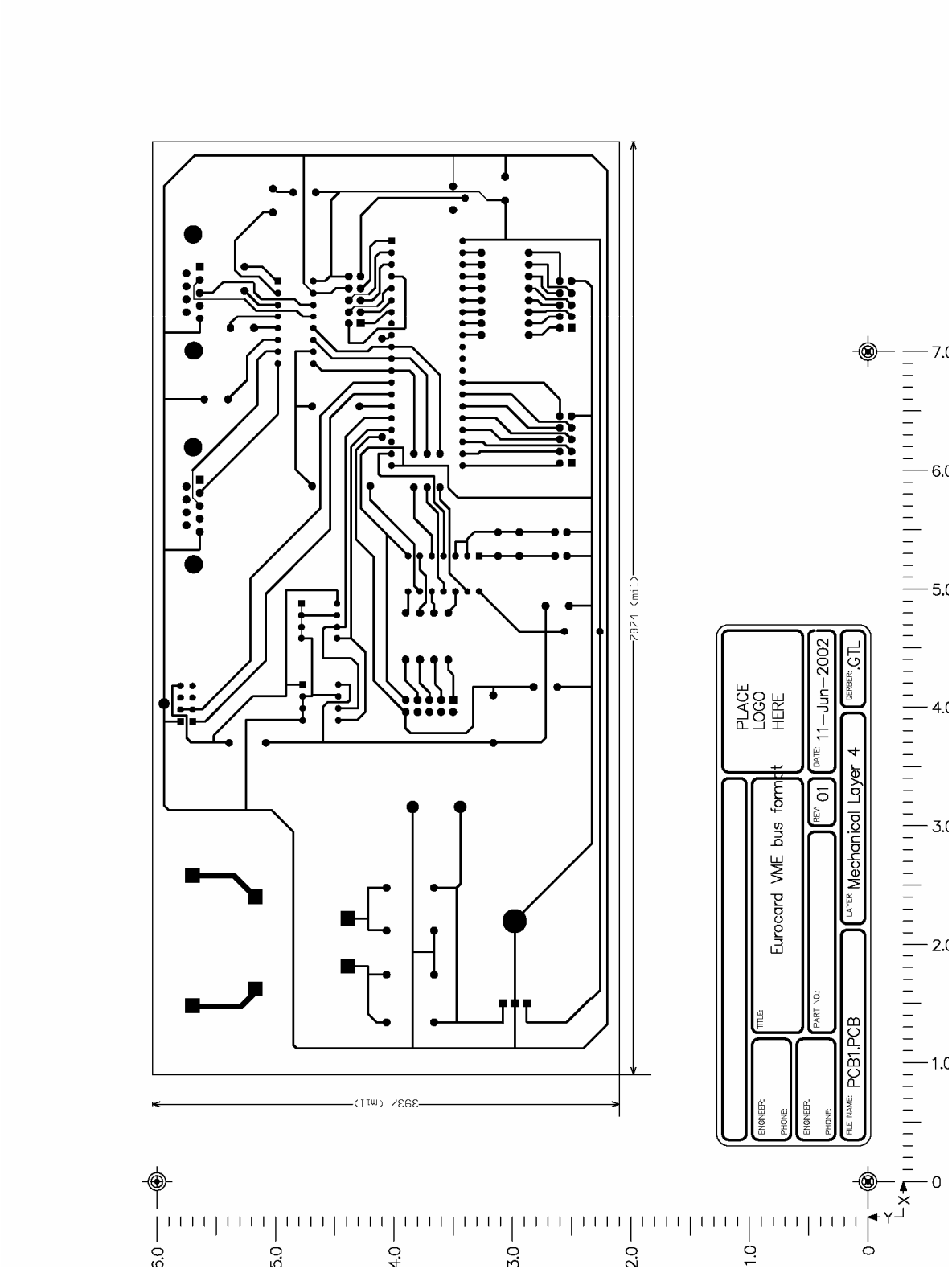
Además, se ha introducido un fusible de seguridad, y se ha usado un interruptor de alimentación de dos vías, usando una de ellas para abrir y cerrar el paso de la alimentación alterna a 220V, y el otro para abrir y cerrar simultáneamente el paso de la señal de alimentación regulada de 5V. De esta forma se han evitado transitorios de alimentación que conducían a errores en la memoria flash y eeprom del microcontrolador.

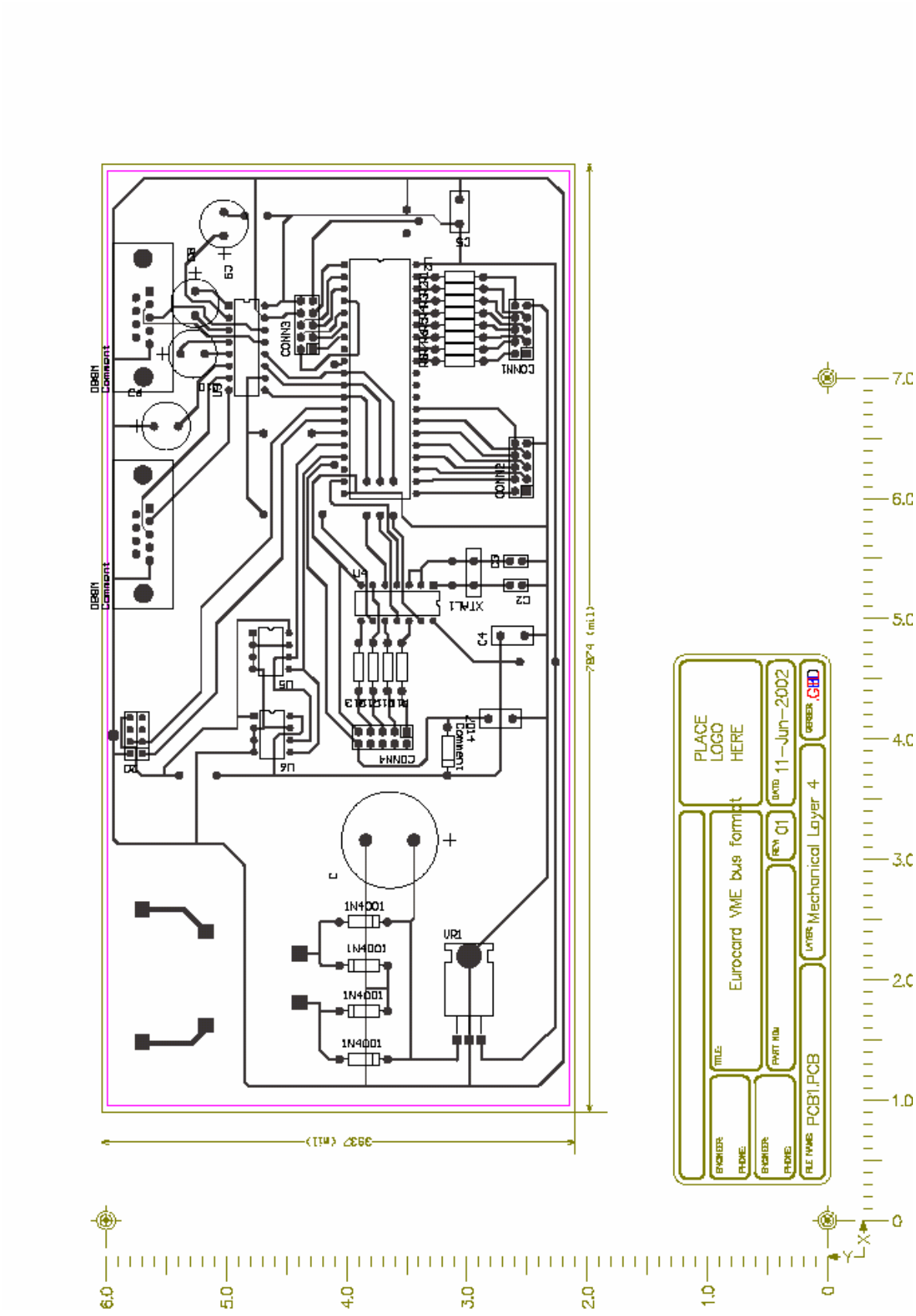
4.3.3.9. Esquema final

En las cuatro siguientes páginas se muestran el esquema y el fotolito final que se han usado para el prototipo del entrenador de Algorítmez. Todas las partes del sistema se han comentado ya en las secciones anteriores por separado.


Microcontrolador central, LCD y LEDS y SWITCHS para Algoritmez
Fuente de alimentacion 220V ac - 5V dc
Oscilador a cristal
Entrenador de Algoritmez
Document Number
Monday, December 03, 2001
TRANSFORMER 220V-12V AC
LCD_16x2
AT90S8515
SW DIP-8
LEDS D1-D8
R1-R8 1K
+Vreg
+5
RESET
RX
TX
RX2
TECL_DAT
TECL_CLK
TX2
SCL
SDA
RELOJ
74HC04
7.3728 MHz
100K
22pF
10K
100uF
100nF
1N4001
7805
220V AC
FUSE
SW1
POT1
Ajuste contraste


Interfaz serie RS-232
COM 2
DB9 MACHO
COM 1
DB9 MACHO
MAX232
U4
C6 1uF
C7 1uF
C8 1uF
C9 1uF
RX2
TX2
TX
RX
R12 1K
R13 1K
R14 1K
R15 1K
LEDS D13
LEDS D14
LEDS D15
LEDS D16
U3C
U3D
U3E
U3F
Circuiteria de reset
PULSADOR RESET
D17 DIODE
C10 220nF
RESET
Interfaz teclado PS-2
JP1
PS2 HEMBRA (AL TECLADO)
TECL_CLK
TECL_DAT
Bancos de memoria
U5
U6
24LC512 - 64Kb EEPROM SERIE
Title
Entrenador de Algoritmez
Document Number
Rev 1.1
Size A
Date: Wednesday, March 20, 2002
Sheet 2 of 2


7374 (mil)
3937 (mil)
PLACE
LOGO
HERE
TITLE:
Eurocard VME bus format
ENGINEER:
PHONE:
ENGINEER:
PHONE:
PART NO.:
REV: 01
DATE: 11-Jun-2002
FILE NAME: PCB1.PCB
LAYER: Mechanical Layer 4
GERBER: .GTL
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
X
Y


PLACE
LOGO
HERE
Eurocard VME bus format
11-Jun-2002
Mechanical Layer 4
PCB1.PCB
1N4001
1N4001
1N4001
1N4001
VR1
CONN3
CONN1
CONN2
CONN4
XTAL1
U5
U6
C1
C2
C3
C4
C5
C6
C7
C8
C9
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
Comment
DRAW
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0

4.3.4. Protocolo de comunicaciones

El entrenador tiene unas necesidades de comunicación, ya que se pretende enviar un programa para realizar la emulación del mismo y se debe proveer de los medios necesarios para una monitorización en tiempo real del seguimiento del programa, así como incluso su modificación sobre la marcha. Para conseguir esto es necesario definir formalmente un protocolo de comunicaciones fiable entre el PC (o cualquier otro sistema de control que se construya) y en el entrenador. De esta forma, se van a dividir las comunicaciones en tres capas lógicas, estudiando cada una por separado para, conjuntamente, conseguir el objetivo de unas comunicaciones punto a punto fiables:

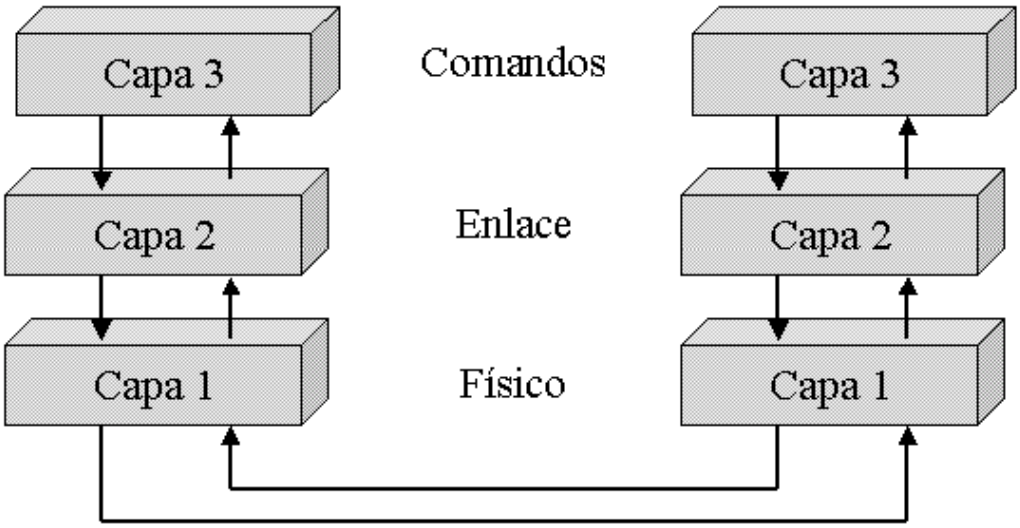


FIGURA 28. Capas de la comunicación

4.3.4.1. Capa física

El propósito de esta capa es la de definir el interfaz físico que permite conectividad a nivel de bits entre los dos extremos. La conexión física entre ambos sistemas se realiza sobre un cable null-modem y niveles de señal según el estándar RS-232, es decir, un puerto serie estándar de PC. Para la comunicación sólo se utilizan las señales RX, TX y GND del puerto serie, por lo que el cable null-modem puede constar solamente de tres hilos, intercambiando las señales RX y TX entre los dos extremos:

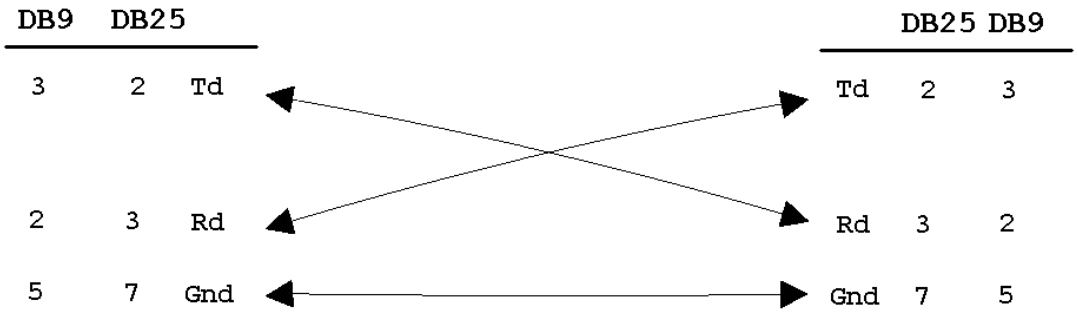


FIGURA 29. Cable null-modem

En este nivel los datos se envían a nivel de bits, agrupados en bytes según los siguientes parámetros:

- Un bit de inicio.
- Bits de datos: 8
- Bits de parada: 2
- Sin bits de paridad.
- Velocidad de bit: 115.200 bps
- LSB primero.

4.3.4.2. Capa de enlace

El propósito de esta capa es la de proporcionar **fiabilidad** al enlace: Detecta fallos en los datos enviados y provee de un mecanismo básico para la repetición, usando un sencillo protocolo de ACK/NACK y *timeouts*, evitando duplicados.
Tanto en PC como el entrenador, mantienen para las funciones de esta capa, buffers separados para transmisión de tramas y para recepción. De esta forma siempre se puede retransmitir la última trama enviada si el otro extremo detecta un error y solicita una repetición (variable ULT_TX en los diagramas).
Otro punto muy importante del protocolo es que el PC será siempre quién envíe una trama al entrenador y éste debe contestar siempre, no pudiendo nunca ser éste quién envíe una trama sin que el PC se lo pida.

El formato de trama de esta es el siguiente:

1 byte	1 byte	1 byte	(variable)	1 byte
CABECERA	**E 1 0 0 0 0 0 0** **INDICADORES**	**LONGITUD**	**MENSAJE**	**LRC**

FIGURA 30. Formato de trama

- **Cabecera**: Es un byte usado para ayudar en la sincronización de trama. Permite asegurar que el byte en cuestión es el comienzo de una trama, aunque siempre queda una probabilidad de error de confundir un byte de mitad de una trama con el comienzo de ésta, al coincidir el byte con el valor esperado de la cabecera. Si se comete un error de este tipo, hay más mecanismos para detectarlo, como el LRC y el timeout entre bytes.
  Este campo debe tener siempre el valor de 0x45 si la trama se envía del PC al entrenador, y de 0x54 si se envía en sentido contrario.

- **Indicadores**: De este byte sólo se usa el bit 7, debiendo estar el bit 6 siempre a 1 y los demás a 0. El bit 7 se usa como indicador de error en LRC para tramas de NACK. Es decir, si uno de los dos extremos detecta un error de LRC en una

trama recibida, devuelve una trama con este bit a 1 para solicitar una retransmisión.

- **Longitud**. Indica la longitud en bytes del campo mensaje, pudiendo ser cero.

- **Mensaje**. Espacio reservado para los datos que la capa superior quiera transmitir, pudiendo ser de longitud variable o no existir siquiera si la longitud es de cero.

- **LRC** (Linear redundance check). Es un byte de comprobación de integridad de toda la trama. Se forma realizando la operación XOR de todos los bytes de la trama, desde la cabecera hasta el último byte del mensaje, de forma que al hacer el XOR de todos los bytes de la trama incluyendo el LRC, el resultado debe ser de cero si no han ocurrido errores. Siendo un método de detección de errores no demasiado estricto, es suficiente para un enlace por cable serie a baja velocidad como es el caso. En la práctica, durante el envío de miles de tramas, nunca se ha detectado un solo fallo de LRC en ninguna de ellas, con lo que demuestra ser un método suficiente para tan baja probabilidad de error, además de ser de implementación rápida y sencilla en el microcontrolador.

Debido a la limitación de almacenamiento de la trama en el lado del entrenador (memoria RAM limitada), se ha establecido un límite máximo de tamaño para una trama completa, incluyendo cabecera, LRC, etc. de 44 bytes. Se ha elegido este valor por ser superior a la trama más grande usada, la de grabación de EEPROM externa:

- Cabecera = 1
- Indicador = 1
- Longitud = 1
- Numero de banco =1
- Dirección = 2
- Numero de bytes =1
- Datos = 32 máximo
- LRC = 1

TOTAL = 40 bytes

Más un margen de otros 4 bytes, para posibles ampliaciones futuras o nuevos comandos de mayor longitud aún. Los detalles de cada comando y sus campos se explican en el siguiente apartado.

Sobre los asentimientos positivos y negativos, a cada comando enviado al entrenador, la respuesta correcta supone un asentimiento positivo de recibo (ACK) intrínseco, distinto según el comando enviado. Sin embargo la trama de asentimiento negativo de recibo (NACK) tiene una estructura fija, ya que el campo de datos permanece vacío: Un NACK del entrenador es $54 C0 00 94, es decir:

Cabecera	Indicadores	Longitud	LRC
0x54	0xC0	0x00	0x94

Mientras que un NACK del PC al entrenador es $45 C0 00 85:

Cabecera	Indicadores	Longitud	LRC
0x45	0xC0	0x00	0x85

Donde el bit 7 (NACK) de los indicadores está activado indicando fallo de LRC y solicitud de reenvío.

El protocolo sólo permite en cada momento estar a la espera de un ACK, luego sólo se pueden enviar las tramas de una en una, esperando a la confirmación de la anterior para enviar la siguiente, usando además un *timeout* para detectar problemas en la comunicación. Esto es así por la corta distancia entre los dos equipos, lo que hace innecesario el uso de protocolos de ventana más complejos y con más requisitos de memoria difícilmente implementables en el microcontrolador elegido.

Los tiempos de timeout en cada caso son:

- Del programa “Comunicador”: El timeout de espera de respuesta depende del tipo de instrucción, ya que algunas llevan mas tiempo en ejecutar que otras. Por defecto es de 1ms. Tras cumplirse, vuelve a enviar la trama si no se ha llegado al límite de intentos.

- Del entrenador: El timeout en este caso es de espera entre dos bytes consecutivos, siendo su valor de 4.4 ms. Si se cumple, se desecha la parte de trama recibida hasta el momento y no se responde nada.

En cuanto al número de reenvios, en el lado del PC se reenvía una trama un máximo de 3 veces si se reciben respuestas NACK o se cumple el *timeout*. En caso de que el envío fuese una trama NACK y se reciba una trama NACK del entrenador o se cumpla el *timeout* , se aborta la transmisión al no ser posible saber con seguridad si el entrenador ha recibido correctamente la trama y evitar posibles duplicados.
Es decir, en caso de un resultado positivo por parte de la capa de enlace se tiene la seguridad de que la trama ha sido recibida por el entrenador, pero en caso de recibir una situación de error, esto no implica necesariamente (aunque sí es lo normal) que la trama no haya sido recibida correctamente por el otro extremo. En cualquier caso la probabilidad de que esto ocurra es despreciable, dado el régimen de fallos de un enlace por cable de tan poca distancia.

Por el lado del entrenador, no existe límite de reenvíos y siempre vuelve a enviar la última trama que transmitió en caso de recibir una trama NACK del PC.

A continuación se muestran algunos ejemplos de comunicaciones y los procedimientos usados en caso de errores. Se han indicado las peticiones a la capa de enlace, a la que se le pide que envíe un comando “Petición (cmd)” y ésta responde con un valor “Ok(resp)” o “Error” según se haya comprobado que el otro extremo ha respondido correctamente o no. En el lado del entrenador, esta capa solo informa a la capa superior de la llegada de un comando correcto.

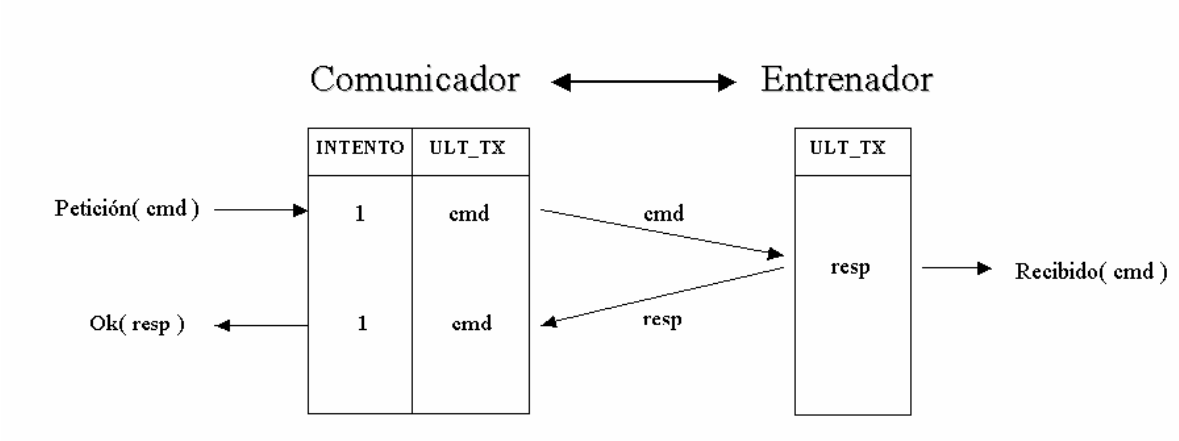


FIGURA 31. Caso normal de transmisión sin problemas

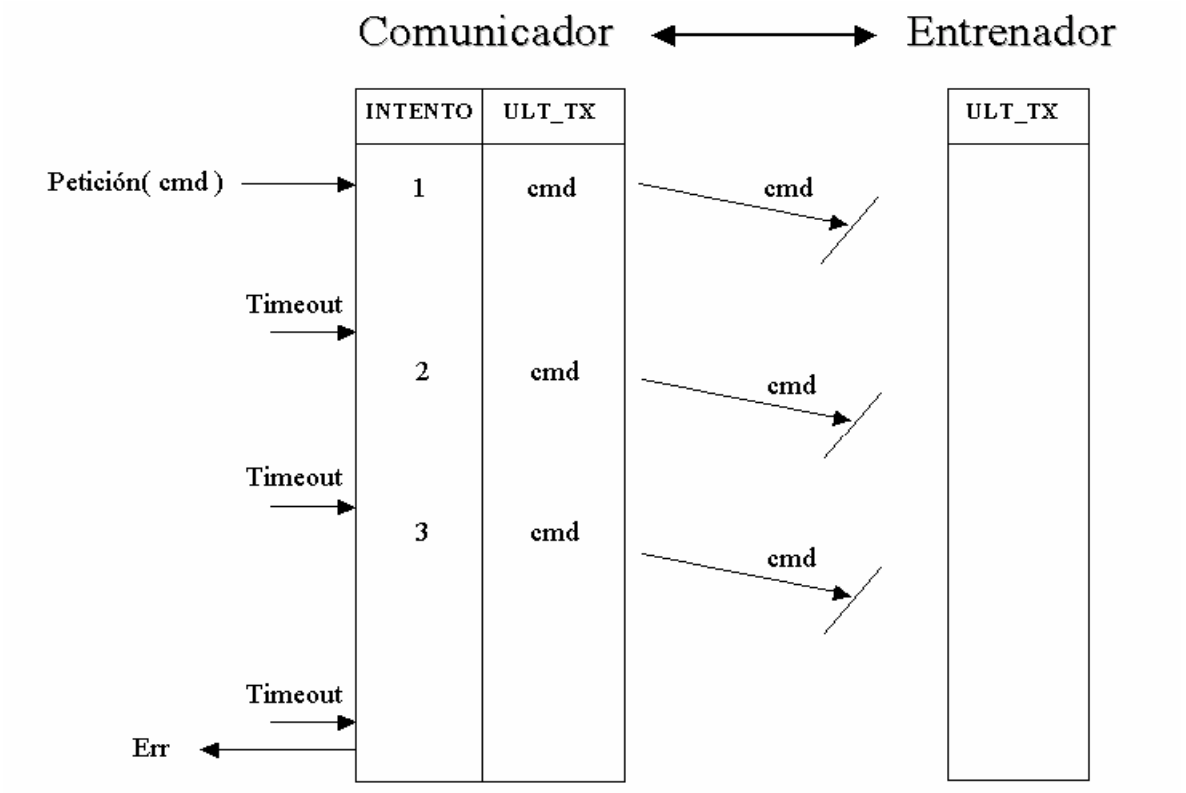


FIGURA 32. Caso de desconexión continua en el enlace físico

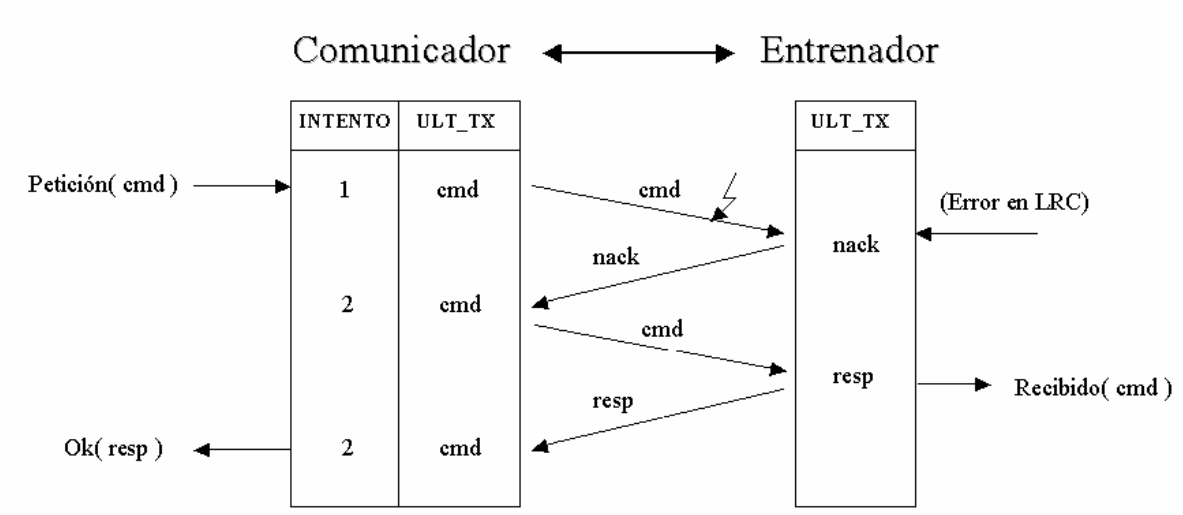


FIGURA 33. Caso de error aislado en la transmisión PC a Entrenador

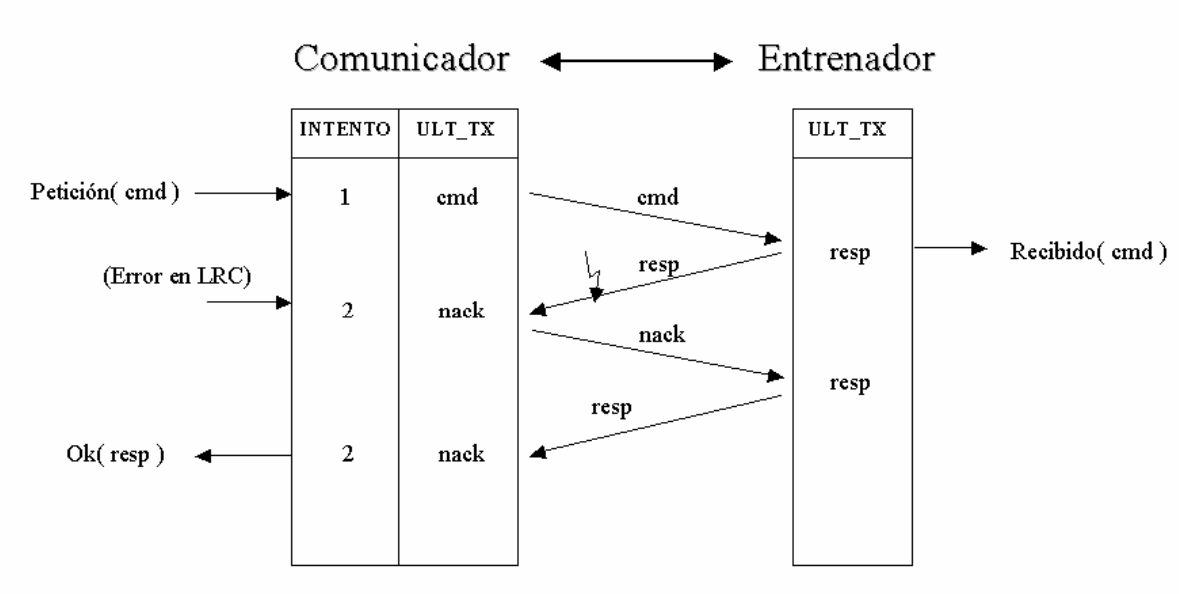


FIGURA 34. Caso de error aislado en la transmisión Entrenador a PC

Se ve como el uso de una trama NACK por parte del comunicador junto con el uso de la variable ULT_TX en el entrenador, permiten recuperar la respuesta del entrenador sin que se produzca un error de duplicados.

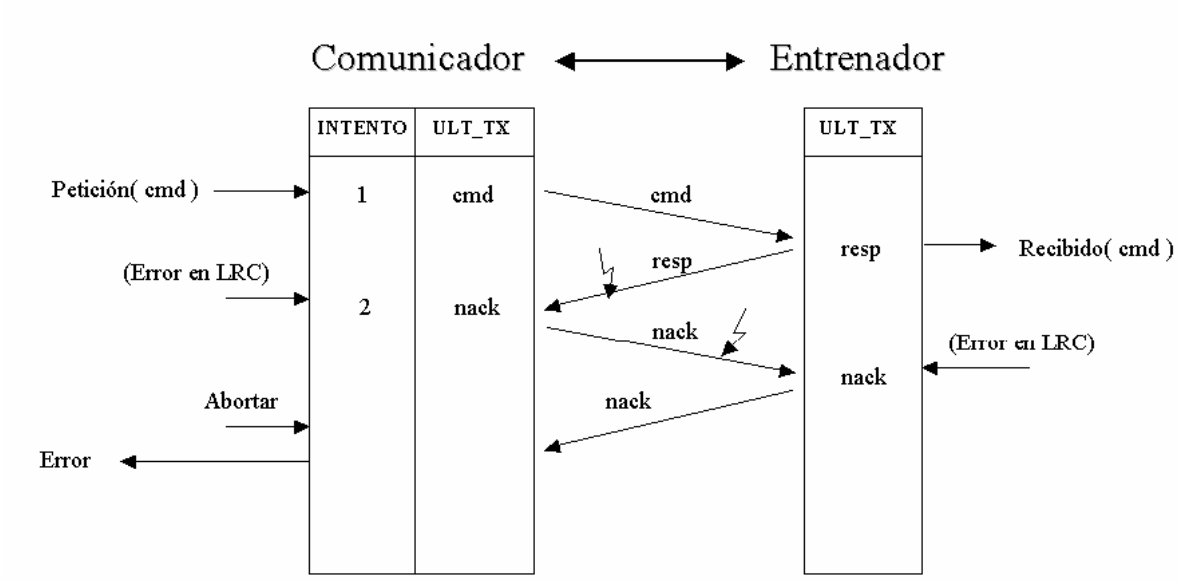


FIGURA 35. Posible caso con dos errores aislados

En este último ejemplo se ve un caso donde, aún cuando el entrenador recibe correctamente la trama, la capa de enlace en el lado del PC indica un error en la transmisión. Como se dijo más arriba, este caso en la práctica es muy improbable debido a las características del medio de transmisión.

4.3.4.3. Capa de comandos

Esta capa se ocupa del formato del contenido del campo de datos "mensaje" de la capa 2. En el sentido del PC al entrenador, este campo de datos se rellena con comandos, y en las respuestas del entrenador al PC se rellena con las respuestas a los comandos.
Siempre debe constar de al menos un byte, que identifica el número de comando que se envía. Para las respuestas, este byte es el número de comando con el bit superior a uno. Por ejemplo, si se envía un comando 0x10, en el mensaje del PC al entrenador el primer byte de los datos es un 0x10 y en la respuesta es un 0x90 = 0x10 | 0x80. El resto de los bytes y su cantidad dependen de cada comando en concreto.

A continuación se muestra una tabla con todos los comandos implementados en el entrenador, detallando a continuación su funcionamiento y parámetros uno por uno:

Número de comando	Número de respuesta asociado	Descripción
0x10	0x90	Comando NOP / Eco.
0x20	0xA0	Escritura en EEPROM externa.
0x21	0xA1	Lectura de EEPROM externa.
0x22	0xA2	Escribir registros de Algorítmez.
0x23	0xA3	Leer registros de Algorítmez.
0x25	0xA5	Leer puertos de E/S de Algorítmez.
0x30	0xB0	Escribir bytes en EEPROM interna.
0x31	0xB1	Leer bytes de EEPROM interna.
0x40	0xC0	Petición de estado del entrenador.
0x41	0xC1	Iniciar ejecución continua.
0x42	0xC2	Parar ejecución continua.
0x43	0xC3	Ejecutar un paso.
0x44	0xC4	Reset de máquina virtual Algorítmez.

TABLA 16. Lista de comandos para entrenador

En los esquemas siguientes, el número entre paréntesis de la descripción de cada campo representa la longitud en bytes de ese campo:

COMANDO 0x10: Comando NOP (O de eco)

Formato del comando:

Nº comando (1)
0x10

Formato de la respuesta:

Nº respuesta (1)
0x90

Función: El entrenador no realiza acción alguna al recibir este comando, salvo como con todos, responder con una trama de respuesta. El uso de este comando es útil para comprobar si el entrenador está operativo y para verificar que la línea de comunicación y configuración del puerto serie son correctos.

COMANDO 0x20: Escritura en EEPROM externa

Formato del comando:

Nº comando (1)	Nº de banco (1)	Dirección (2)	Nº de bytes (1)	Datos (Var)
0x20	0x00 / 0x01	0x0000 – 0xFFFF	0x00 – 0x20	D[0]…D[N]

Formato de la respuesta:

Nº respuesta (1 byte)
0xA0

Función: Permite grabar una secuencia de desde 1 hasta 32 bytes en una dirección dada del chip de memoria seleccionado (0 o 1) de EEPROM externa. Esto puede usarse para:

- La programación del programa de Algorítmez: Al grabar el contenido en el banco 1 de la imagen RAM de 64Kb a emular.
- Modificación durante depuración de la RAM: Escribiendo en el banco 0 (RAM actual) durante la depuración de un programa.

COMANDO 0x21: Lectura de EEPROM externa

Formato del comando:

Nº comando (1)	Nº de banco (1)	Dirección (2)	Nº de bytes (1)
0x21	0x00 / 0x01	0x0000 – 0xFFFF	0x00 – 0x20

Formato de la respuesta:

Nº respuesta (1 byte)	Datos (Var)
0xA1	D[0]...D[N]

Función: Leer un bloque de hasta 32 bytes de los bancos 0 o 1 de EEPROM externa. Leer del banco 0 durante la depuración es útil para la inspección de las variables de memoria del programa emulado.

COMANDO 0x22: Escritura de los registros

Formato del comando:

Nº comando (1)	.0 a .15 (32)	RE (2)
0x22	D[0]…D[32]	D[0]..D[1]

Formato de la respuesta:

Nº respuesta (1 byte)
0xA2

Función: Cambia el valor de los 16 registros de Algorítmez (.0 ... .15) y del registro de estado (RE) por los nuevos valores enviados. De cada registro se envía primero el byte alto y después el bajo.

COMANDO 0x23: Lectura de los registros

Formato del comando:

Nº comando (1)
0x23

Formato de la respuesta:

Nº respuesta (1 byte)	.0 a .15 (32)	RE (2)
0xA3	D[0]…D[32]	D[0]..D[1]

Función: Lee el estado de los 16 registros de Algorítmez (.0 ... .15) y del registro de estado (RE). Al igual que en el comando anterior, se envían primero el byte alto de cada registro y a continuación el bajo.

COMANDO 0x25: Lectura de los puertos de E/S de Algorítmez

Formato del comando:

Nº comando (1)
0x25

Formato de la respuesta:

Nº respuesta (1 byte)	Estado de puertos (9bytes)
0xA5	D[0]…D[9]

Función: Lee el estado de los 16 registros de Algorítmez (.0 ... .15) y del registro de estado (RE). Al igual que en el comando anterior, se envían primero el byte alto de cada registro y a continuación el bajo.

COMANDO 0x30: Escribir en EEPROM interna

Formato del comando:

Nº comando (1)	Bytes (15)
0x30	D[0]..D[14]

Formato de la respuesta:

Nº respuesta (1)
0xB0

Función: Permite grabar el registro de configuración en la EEPROM interna del entrenador. El significado de estos datos depende de su posición en el registro, cuya estructura se ve en el punto 4.3.5.2. Con este comando se guardan dos copias del registro de configuración: Una en la posición habitual y otra en la posición de copia de seguridad (También descrita en 4.3.5.2.).

COMANDO 0x31: Leer de EEPROM interna

Formato del comando:

Nº comando (1)
0x31

Formato de la respuesta:

Nº respuesta (1)	Bytes (15)
0xB1	D[0]..D[14]

Función: Permite leer el registro de configuración de la EEPROM interna del entrenador. El significado de estos datos depende de su posición en el registro, cuya estructura se ve en el punto 4.3.5.2.

COMANDO 0x40: Petición de estado

Formato del comando:

Nº comando (1)
0x40

Formato de la respuesta:

Nº respuesta (1)	Estado (1)							
0xC0	0	0	0	0	0	0	R/S	E/D

Función: Obtiene el byte de estado del entrenador. De éste sólo se usan los dos bits inferiores:

- Bit 0 – E/D: Indica que el entrenador se ha iniciado en modo de sólo ejecución (si está a 1) o en modo de depuración (si está a 0)
- Bit 1 – R/S: Indica que el entrenador está en modo de ejecución continua (si está a 1) o está parado (si está a 0). Esto último sólo es posible en modo depuración.

COMANDO 0x41: Inicia modo ejecución continuo

Formato del comando:

Nº comando (1)
0x41

Formato de la respuesta:

Nº respuesta (1)
0xC1

Función: Inicia el modo de ejecución continuo del entrenador. Esto sólo se aplica al modo depuración, ya que en el modo ejecución siempre se está en ejecución continua. Si ya se estaba en modo ejecución continua, el comando no tiene efecto ni da lugar a ningún error. Es equivalente a pulsar la tecla F10 en el entrenador.

COMANDO 0x42: Parar modo ejecución continuo

Formato del comando:

Nº comando (1)
0x42

Formato de la respuesta:

Nº respuesta (1)
0xC2

Función: Para el modo de ejecución continuo del entrenador. Si no se está en modo de depuración, o ya se está en modo de parada, el entrenador ignora este comando, aunque responde correctamente. Es equivalente a pulsar la tecla F9 en el entrenador.

COMANDO 0x43: Ejecutar un paso

Formato del comando:

Nº comando (1)
0x43

Formato de la respuesta:

Nº respuesta (1)
0xC3

Función: Ejecuta una sola instrucción de Algorítmez. Este comando sólo tiene efecto si el entrenador está en modo depuración y parado. Se actualiza el estado en el display del entrenador tras la ejecución de la instrucción. Es equivalente a pulsar F8 en el entrenador.

COMANDO 0x44: Reset de máquina virtual Algorítmez

Formato del comando:

Nº comando (1)
0x44

Formato de la respuesta:

Nº respuesta (1)
0xC4

Función: Produce un reset de la máquina virtual Algorítmez: Reinicio de los registros a cero, CP y PP a valores iniciales, puertos a sus estados por defecto, y restauración de la imagen de memoria RAM. Aunque el entrenador responde inmediatamente a éste comando, mientras se está reconstruyendo la imagen de RAM (lo que tarda unos segundos) no responde a ningún otro comando. Un método para saber cuando vuelve a estar operativo (usado en el comunicador) es enviar periódicamente comandos NOP hasta que el entrenador responda.

4.3.5. Firmware del entrenador

4.3.5.1. Diseño general

Se ha diseñado el firmware del microcontrolador AT90S8515 de forma que éste se comporte como el microprocesador Algorítmez, siempre que se usen los periféricos y conexión a memoria descritos en el punto 4.3.3.
Además se le ha añadido un entorno de depuración integrado de habilitación opcional, de forma que sea posible depurar programas Algorítmez sin hacer uso del programa "Comunicador" en el PC.

Para conseguir la mayor eficiencia posible, se ha hecho uso en todas las situaciones posibles de las interrupciones de que dispone en microcontrolador, de forma que ciertas tareas se realicen de forma "paralela" al hilo principal del emulador, comunicándose estos a través de variables y buffers en memoria.

El entrenador tiene 3 modos de funcionamiento distintos:

- Modo ejecución. En este modo el entrenador se comporta solamente como el microprocesador Algorítmez, mostrándose en pantalla el estado de la pantalla virtual de Algorítmez y ejecutando sin interrupción el programa almacenado en la RAM de Algorítmez.

- Modo depuración, parado. Esta estado también es llamado en el programa "menú principal" o "pantalla principal". No se ejecutan continuamente instrucciones, sino que se muestra en pantalla el valor actual del CP y la próxima instrucción a ejecutar, en forma desensamblada y en hexadecimal:

Es desde esta pantalla desde donde se tiene acceso a todos los menús del entrenador, según las teclas pulsadas en el teclado del entrenador. Los menús se detallan en el punto 4.3.5.2.

- Modo depuración, en ejecución. En este modo se muestra en el display el contenido la pantalla del Algorítmez, y se ejecutan continuamente instrucciones del programa. Se puede parar la ejecución pulsando F9 (o mediante comando desde el Comunicador) para volver al estado depuración y parado.

Independientemente del modo de funcionamiento del entrenador, siempre se atiende a los comandos de control recibidos por el puerto COM1.

Existe una variable de un byte que representa el estado del entrenador, guardada en la RAM del microcontrolador y que define el comportamiento del mismo entre los tres modos descritos arriba:

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
(No usado)						0= Parado (en depuración) 1= Ejecución continua (en depuración)	0=Modo depuración 1=Modo ejecución

FIGURA 36. Formato del byte de estado del entrenador

El diagrama de flujo general resumido del entrenador queda como se ve en la siguiente página.

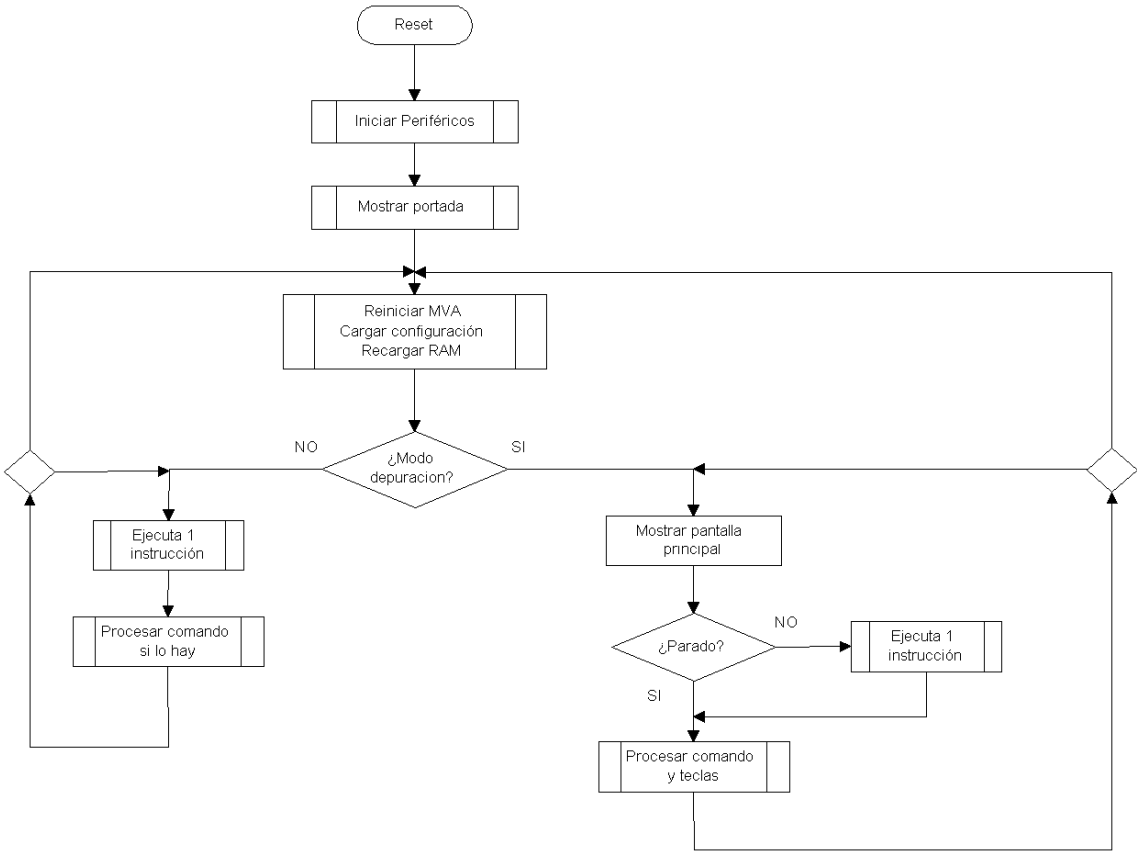


FIGURA 37. Diagrama de flujo resumido del entrenador

Donde por claridad se han omitido los menús a los que se puede acceder desde la pantalla principal en modo depuración.

Los comandos recibidos del PC se van almacenando por interrupciones, marcando una variable cuando el comando se ha recibido completamente. De esta forma, el diagrama de flujo principal solo tiene que comprobar periódicamente como se ve, si hay o no un comando preparado para procesarlo. Las teclas pulsadas en el teclado se almacenan siguiendo el mismo método de recibirlas por interrupciones y activar una señal en una variable.

4.3.5.2. Sistema de menús

A continuación se muestra un esquema con todos los menús implementados en el entrenador. Se parte de la pantalla principal (siempre en modo depuración, ya que en modo ejecución no se puede acceder a ningún menú) y se muestra sobre los enlaces la tecla con la que se accede a cada menú. En la mayoría de los casos la tecla escape (ESC) sirve para salir del menú actual y volver al anterior:

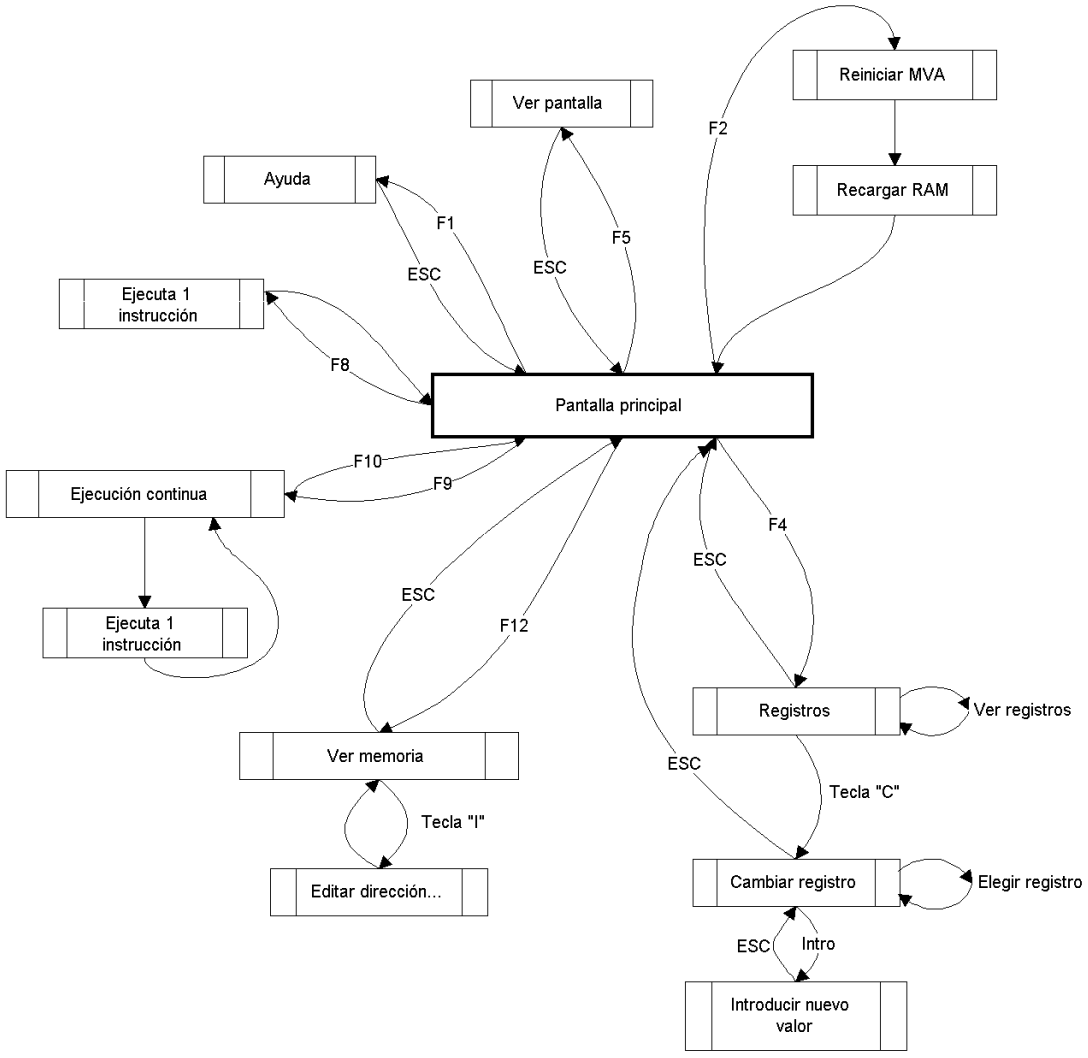


FIGURA 38. Diagrama de flujo del sistema de menús del entrenador

- Pantalla principal: Como ya se vio mas arriba, en estado de depuración/parado aquí se muestra desensamblada la siguiente instrucción a ejecutar.

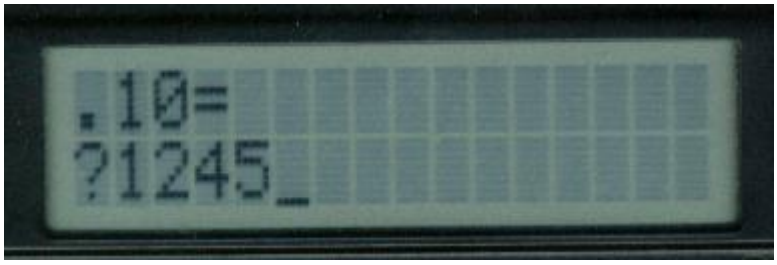
- Ayuda: Pulsando F1 se muestra la pantalla de ayuda, donde pulsando las teclas de flecha arriba y flecha abajo se hace scroll por las líneas del texto, que enumera las teclas con las que se entra a cada menú:

- Reset de MVA (Máquina virtual Algorítmez): Pulsando F2 se reinicia el estado de Algorítmez y se recarga la memoria del banco 0 con el contenido del banco 1, es decir, con la copia original del programa a ejecutar:

- Ver/editar registros: Pulsando F4 se muestra el visor de registros, donde se puede ver el listado de los 16 registros y su contenido (en hexadecimal), además del contenido del registro de estado, este último en binario y con una leyenda del significado de cada bit. Para moverse por la lista hay que usar las teclas de flecha arriba y flecha abajo:

Pulsando la tecla “C” en este menú se entra en el menú de edición de registros, donde se permite cambiar el contenido de cualquiera de ellos. Aparece una lista de los registros donde con las flechas de arriba y abajo se selecciona el registro que se quiere cambiar:

Y pulsando Intro sobre este, se nos pide que introduzcamos por teclado el nuevo valor en hexadecimal:



- Ver pantalla de Algorítmez: Pulsando F5 se muestra el contenido actual de la pantalla de Algorítmez, hasta que se pulse ESC para volver a la pantalla principal:

- Ejecutar un paso: Pulsando F8 se ejecuta sólo una instrucción y se muestra la nueva siguiente instrucción desensamblada.

- Iniciar ejecución continua: Pulsando F10 se inicia el modo de ejecución continua. En este modo se muestra el estado de la pantalla virtual de Algorítmez y se ejecutan instrucciones continuamente, hasta que se pulse F9 para parar, o se llegue a un punto de ruptura, o se reciba del comunicador un comando de parar ejecución. Al parar se vuelve a la pantalla principal mostrando la nueva siguiente instrucción a ejecutar.

- Visor de memoria: Pulsando F12 se entra a la vista de memoria:

A pesar de las limitaciones de tamaño del LCD se ha conseguido mostrar los siguientes campos por línea: Dirección, contenido (3 bytes por línea) y contenido ASCII (esos 3 bytes mostrados como caracteres).

En este modo, se muestra el cursor y se puede mover usando las cuatro flechas de dirección, además de las dos teclas de avanzar página y retroceder página. Además, pulsando la tecla “I” se puede ir directamente a una dirección determinada:

4.3.5.3. Uso de la EEPROM interna

Dentro de los 512 bytes de memoria EEPROM de que dispone el microcontrolador, se ha definido un formato de registro de control. Este se almacena en la dirección 0x00 de dicha memoria, y permite que el entrenador mantenga su configuración aunque se desconecte de la alimentación. Aquí es donde se guardan los parámetros de configuración que se reciben del Comunicador. Su formato es el siguiente:

Dirección	Longitud	Descripción
0x00	0x01	Modo de funcionamiento
0x01	0x02	Valor inicial de CP
0x03	0x02	Valor inicial de PP
0x05	0x01	Numero de breakpoints activos
0x06	0x02	Dirección del breakpoint 1
0x08	0x02	Dirección del breakpoint 2
0x0A	0x02	Dirección del breakpoint 3
0x0C	0x02	Dirección del breakpoint 4
0x0E	0x01	Checksum: Suma de los 14 bytes inferiores, ignorando acarreos.

TABLA 17. Formato de registro de configuración en EEPROM

Siempre existe un riesgo con las memorias EEPROM, debido a su tendencia a corromperse bajo ciertas situaciones de transitorios de voltaje, cortes de alimentación durante una escritura, etc...

Para evitar que esto implique una perdida de la información almacenada en este registro, se ha declarado un byte de *checksum* para la verificación de la integridad de los datos. Además, siempre que se modifica el registro desde el Comunicador, se guarda una segunda copia de estos 15 bytes (incluido el *checksum* correcto) a partir de la dirección 0x10.

Así, si el simulador de Algorítmez detecta en la comprobación que realiza al iniciarse, un valor de *checksum* incorrecto en el primer registro, sobrescribe este por su copia de seguridad. El método no es infalible, pero proporciona una cierta seguridad de la integridad de la configuración.

4.3.5.4. Interrupciones usadas

Se usan las siguientes interrupciones del microcontrolador:

- INT0, entrada de interrupción externa 0. Programada para generar interrupción con un nivel lógico bajo, está conectada a la línea RX del puerto serie COM2 (previa conversión de los niveles lógicos). Es decir, se usa como interrupción de señalización de un bit de inicio en la recepción de la UART de Algorítmez.

- INT1, entrada de interrupción externa 0. Programada para generar interrupción con un nivel lógico bajo, está conectada a la línea de datos del teclado, con lo que se lanzará una interrupción cada vez que el teclado comience a transmitir un byte, como se describió en el apartado 4.3.3.3.

- Timer1 Overflow. Este temporizador es usado para el timeout de la capa de enlace de comunicaciones, como se explica en 4.3.5.8.

- UART Reception completed. Usada para detectar la llegada completa de un byte por el puerto serie COM1, conectada a la UART hardware del microcontrolador. Por aquí se reciben los comandos que envía el PC.

4.3.5.5. Subsistema de teclado

Se usa la línea de interrupción INT1 conectada a la línea de datos del teclado para generar una interrupción cada vez que el teclado comience a transmitir un byte. La rutina de procesado de esta interrupción, recibe el byte por el bus síncrono del teclado. Como se vio en el funcionamiento de los teclados, estos envían continuamente el scancode de la tecla pulsada hasta que al soltarla envían un 0xF0 seguido del scancode de la tecla soltada.

Ahora bien, para comunicar al resto del programa que se ha recibido una tecla, se usan dos variables en memoria RAM, que forman el buffer de tecla pulsada:

```
; Modulo TECLADO ESTANDAR PC
KEYB_HAY:   .byte 1
KEYB_BUF:   .byte 1
```

Donde en la primera variable se almacena normalmente un 0x00, hasta que se recibe un carácter y se pone a 0x01 guardando además el código ASCII de la tecla pulsada en la segunda variable. Cuando el programa lea el contenido de la tecla pulsada, debe poner también a cero la variable indicadora KEYB_HAY. Esto se realiza siempre con las interrupciones inhibidas, para evitar que se pierda una nueva tecla que se guarde en esta variable justo antes de borrarla.

Con todo esto, esta rutina realiza las siguientes tareas:

- Si recibe un scancode correcto, diferencia entre si éste es estándar o extendido (prefijo 0xE0, con lo que se espera a recibir un nuevo carácter) y mira en una tabla su correspondiente código ASCII. Si el buffer de tecla pulsada esta libre, almacena esta tecla en él. Si ya esta lleno, la nueva tecla se pierde.
- Si recibe un código de tecla soltada 0xF0 + scancode, lo ignora.
- Si se recibe algún error en un bit de paridad, se ignora ese scan-code.
- Si se recibe un scan-code que no está en la tabla de scancodes/ASCII, se ignora.

Debido a las limitaciones de espacio de memoria de programa (donde están las tablas de conversión scancode/ASCII), no se ha podido implementar el funcionamiento de la tecla “Mayúsculas” ya que esto implica dos tablas completas, una para cada estado de la tecla mayúsculas. Por lo tanto, todas las teclas de letras se corresponden con sus códigos ASCII en mayúsculas y no se pueden escribir los caracteres que normalmente se forman con el uso de esta tecla: paréntesis, signo de dollar, barra inclinada, comillas, etc...
Tampoco se procesan las teclas de bloque de mayúsculas ni numérico.

A las cuatro teclas de dirección se han asignado códigos “ASCII” que se corresponden con las flechas horizontales y verticales en el mapa de caracteres del LCD.

4.3.5.6. Subsistema de display

Para el uso del display se han creado una serie de rutinas que facilitan multitud de operaciones sobre este, aunque todos se basan en dos rutinas básicas:

- `LCD_SEND_R16_DATA`
- `LCD_SEND_R16_INST`

Las dos envían el byte en el registro r16 del microcontrolador por el bus del LCD, diviéndolo en dos nibbles como se vio en 4.3.3.2. La diferencia es que una rutina lo manda como dato y la otra como instrucción, lo que se refleja en la señal RS del LCD.
Tras enviar el byte, se inicia el ciclo de espera, para asegurar que el display ha terminado la operación. Para esto se entra en un bucle que lee continuamente el byte de estado del LCD (En 4.3.3.2. se detalla como se realiza esto), hasta que el bit 7 de éste, el “Busy Flag” esté a 0, lo que indicará que ya ha terminado de procesar la operación.
Con esto se ahorra mucho tiempo al no realizar bucles de espera fijos, sino que se espera sólo el tiempo requerido por el LCD. Además estos tiempos pueden variar mucho entre distintos modelos, con lo que el método de *polling* es la mejor alternativa.

Algunas de las rutinas que se han implementado para el manejo del LCD son estas:

- LCD_INIT: Reinicia el LCD y lo pone en modo de dos líneas. Configura además los puertos del microcontrolador como salidas para el bus con el display. Además, genera dos caracteres de usuario, la flecha arriba y la flecha abajo, de códigos 2 y 3 respectivamente.
- LCD_SEND_R16_DATA: Directamente sirve para enviar un carácter en la posición actual del cursor.
- LCD_BORRAR: Borra el LCD y coloca el cursor al comienzo.
- LCD_CURSOR_NO_BLINK y LCD_CURSOR_BLINK: Muestra el cursor parpadeando o no.
- LCD_CURSOR_OCULTAR: Oculta el cursor.
- LCD_SET_CURSOR_POS_R16: mueve el cursor a la posición dada en el registro r16. Las posiciones van de 0x00 a 0x0F y de 0x40 a 0x4F para la primera y segunda línea, respectivamente.
- LCD_ESCRIBE_R17_BIN: Escribe el byte en r17 en binario con 8 dígitos.
- LCD_ESCRIBE_R16_HEX: Escribe el byte en r16 en hexadecimal con 2 dígitos.
- LCD_ESCRIBE_CADENA_Z_FLASH: Escribe una cadena de texto terminada en carácter 0x00, apuntada por el puntero Z en memoria FLASH de programa.
- LCD_ESCRIBE_CADENA_Z_RAM: Igual que el anterior, pero el puntero Z apunta a una cadena formada en memoria RAM.
- LCD_R17_ESCRIBE_BCD: Escribe el valor en r17 como numero decimal sin signo.

4.3.5.7. Subsistema emulador de Algorítmez

El estado de la máquina virtual Algorítmez (MVA) emulada en el microcontrolador, esta formado por este conjunto de variables:

- Memoria RAM. Son 64Kb almacenados en el banco 0 de memoria externa I2C.
- Registros. Los 16 registros y el registro de estado, almacenados en memoria RAM del microcontrolador.
- Periféricos. Existen una serie de variables que almacenan los valores de los puertos de cada periférico. Estos se almacenan también en memoria RAM interna del microcontrolador.
- Pantalla virtual de Algorítmez. Formada por un buffer en RAM de 2x16 bytes (2 líneas por 16 caracteres por línea).

Las rutinas principales que se encargan de la MVA en el firmware son las siguientes:

ALG_RESET

Reinicia el estado de los registros y periféricos de la MVA. Además recarga el estado de la RAM con la copia del segundo banco de memoria externa. Para minimizar este tiempo de copia, se van leyendo bloques de 32 bytes de la misma dirección en cada uno de los bancos, se comparan y solo en caso de encontrar alguna diferencia se inicia la escritura del bloque en el banco 0 de memoria. Esto se realiza además para minimizar el número de escrituras innecesarias en la memoria EEPROM que permiten un número alto pero limitado de re-escrituras, no dando garantía el fabricante de que las escrituras tengan éxito si se supera dicho límite, que suele estar en torno a 100.000 escrituras en cada dirección.

ALG_IN_PORT

Permite obtener el valor que se obtendría al realizar una operación de lectura sobre el puerto indicado en el registro "temp2" del microcontrolador, dejando el resultado en el registro "temp". Como algunas lecturas en puertos llevan acarreadas efectos secundarios (como por ejemplo, poner a cero el bit de preparado al leer el puerto de datos del teclado) se usa un registro ("aux") como indicador, si su valor es distinto de cero, de que estos efectos secundarios deben efectuarse. Esto es útil para la ejecución del comando "Obtener estado de puertos" que desde el PC se puede enviar al entrenador. En esta lectura de los puertos, no se pueden producir efectos secundarios, que se producen solamente al ejecutar una instrucción Algorítmez `'IN'` sobre ese puerto. Todos los valores de los puertos se almacenan en la RAM del microcontrolador, excepto el de datos de los switchs, el cual se lee directamente del correspondiente puerto del microcontrolador.

Además, realizar entradas sobre puertos no definidos devuelve cero.

ALG_OUT_PORT

Realiza la operación de sacar el valor en el registro "temp" por el puerto número "temp2". En esta misma rutina se ejecutan:

- Las salidas serie de la UART simulada por software de Algorítmez. Para la temporización de los tiempos de bit se usa el TIMER0 del microcontrolador, programándolo de acuerdo con la velocidad configurada en el puerto de estado de la UART.
- Las salidas a la pantalla. Esto sólo implica la actualización en RAM del buffer de la pantalla virtual de Algorítmez. Además, se actualiza un indicador de refresco de display, que permite en modo de ejecución continua que solo se actualice el display cuando realmente sea necesario. Aquí se implementan las funciones de borrado de pantalla virtual si el carácter a escribir es un 0x00, retorno de carro y avance de línea si el carácter es un 0x0D y *scroll* de línea si el carácter es un 0x01. La posición del cursor de Algorítmez se mantiene también en RAM y se actualiza automáticamente según la operación realizada. Si se escribe más allá del límite de una línea (16 caracteres) se pasa a la siguiente línea automáticamente, incluso realizando *scroll* si es necesario. Se pone el bit de preparado a 0 y el contador de pantalla ocupada a 50, es decir, durante el tiempo de 50 instrucciones el bit de preparado permanecerá a 0.

Las salidas realizadas sobre puertos de estado son convenientemente enmascaradas para que sólo tengan efecto algunos bits. En concreto, sólo se puede escribir el bit de INT (Interrupción permitida) de cada puerto de estado, salvo en la UART en la que además se permite escribir los bits del campo VEL (Velocidad de bit).

ALG_DESEN_DIR_Z

Esta compleja rutina desensambla una instrucción de Algorítmez, colocada en la dirección pasada en ZH:L (registro del microcontrolador) de la memoria RAM emulada, es decir en el banco 0 de EEPROM externa. Se cargan hasta 4 bytes que pueden formar la instrucción, y se analiza según su tipo y modo de direccionamiento, generando en un buffer de la RAM interna (“TEMP_BUFFER”) la cadena con la representación de la instrucción y sus operandos.
Lo primero que se realiza es la búsqueda del opcode y tipo en una tabla de este tipo:

```
...
.db  $4E,"NEG    "
.db  $4F,"ADJ    "

.db  $80,"ADD    "       ; Tipo 2
.db  $81,"ADD.B  "
...
```

Donde se compara el primer byte de la instrucción menos los bits del campo de modo de direccionamiento, con los valores de la tabla, hasta encontrar la cadena correcta que representa el mnemónico de la instrucción. Si se trata de un opcode no usado, se muestra el texto “???” y no se procesan los operandos.
Tras obtener el mnemónico, se analizan las instrucciones dependiendo de su tipo:

- Tipo 0: Ningún operando, luego con el mnemónico ya basta.
- Tipo 1: Colocar el registro usado, o en caso de IN y OUT, los dos registros.
- Tipos 2 y 3: Ambos usan acceso a memoria y en principio se tratan conjuntamente, calculando el modo de direccionamiento y añadiéndolo a la

cadena. Se detectan los modos con CRX=15 para escribirlas en un formato distinto. Por ejemplo, un direccionamiento directo (`LD .0,#1`) se muestra más claro como direccionamiento directo que como direccionamiento de autoincremento (`LD .0 , [.15++]`). Tras esto, sólo si era una instrucción de tipo 2 se añaden la referencia a registro.

En algunos casos es necesario conocer que instrucciones operan sobre 1 o 2 bytes, lo que se ha implementado con una rutina ("`ALG_ES_INSTRUCCION_TAMANO_BYTE`") que busca en su mnemónico la subcadena ".B" para identificar las instrucciones de tamaño byte.

ALG_NUM_BYTES_INST_Z_R0

Esta rutina, al igual que la anterior, analiza una instrucción de Algorítmez situada en la dirección ZH:L de la memoria RAM emulada. En este caso se determina el número de bytes que ocupa la instrucción almacenando el resultado en el registro `r0`. Esta rutina es usada por la rutina de la pantalla principal, donde se muestran los bytes de que se compone la instrucción, para lo que es necesario conocer su número.

ALG_EJECUTA_1_INST

Esta rutina es el núcleo del entrenador de Algorítmez, ya que aquí es donde se realiza la emulación de ejecución para cada instrucción y donde se comprueban y lanzan las interrupciones Algorítmez. En la siguiente página se muestra el diagrama de flujo general de esta rutina:

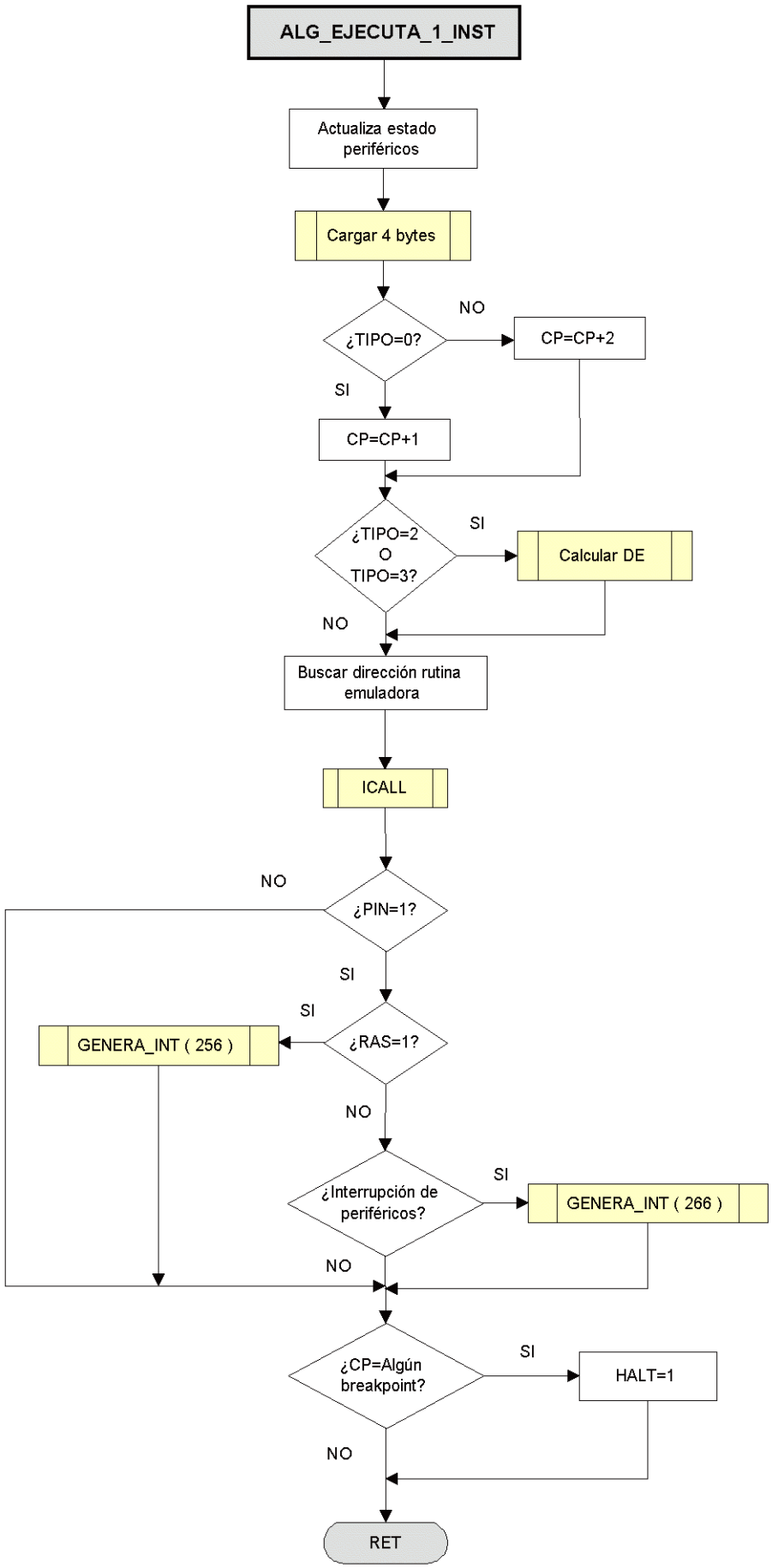


FIGURA 39. Diagrama de flujo para ALG_EJECUTA_1_INST

A continuación se explican algunos de estos pasos:

- "Actualiza estado de periféricos". En esta parte se decrementa el contador de pantalla ocupada y, si este llega a cero, se pone a 1 el bit de preparado del puerto de estado de la pantalla. Además, se lee el estado de los switchs (físicamente el puerto C del microcontrolador) y se compara con el estado en el ciclo de instrucción anterior (almacenada en una variable en RAM). Si se detecta una diferencia, se marca el bit de preparado del puerto de estado de los switchs.

- "Cargar 4 bytes". El porqué de cargar los 4 bytes al principio aunque luego no se usen algunos de ellos, está en el protocolo usado por las memorias I2C. Si solo se lee inicialmente uno o dos bytes, la sobrecarga de protocolo (100 · número de bytes de datos leídos / número de intervalos de bytes usados en total) sería del 80% o del 67% respectivamente. Leer otro byte extra en caso necesario conllevaría una sobrecarga de protocolo del 50% (en esta segunda operación, ya en modo de lectura secuencial) o del 33% en caso de otros 2 bytes. Por esto y para simplificar las rutinas de obtención de operandos, se ha elegido leer secuencialmente los 4 bytes al comienzo, con lo que la sobrecarga del protocolo total es de 50%, valor bajo comparado con los anteriores casos.

- "Calcular DE". En esta subrutina ("`ALG_CALC_DE_A_X`") se calcula la dirección efectiva del operando de memoria de la instrucción, dejándola almacenada en el puntero del microcontrolador XH:L. Esta misma rutina realiza todos los auto-incrementos y auto-decrementos en los registros necesarios. Para realizar esto es de nuevo necesario conocer en algunos casos el tamaño del operando (1 o 2 bytes), por lo que de nuevo se usa la subrutina "`ALG_ES_INSTRUCCION_TAMANO_BYTE`"

- "Busca dirección de rutina emuladora". Esta parte es de la más importantes, ya que para cada instrucción se ha escrito una pequeña rutina que simula su ejecución, y en este punto se busca la dirección en memoria de programa del microcontrolador donde está esta rutina, para poder ejecutarla con la instrucción "ICALL" (Indirect Call, llamada indirecta a rutina apuntada por ZH:L).

Se ha colocado al final de la memoria de programa del microcontrolador, una tabla colocada en una dirección conocida y que contiene las direcciones de todas las rutinas de emulación de instrucciones, ordenadas por orden de tipo (0,1,2 y 3) y dentro de cada tipo por opcode (0, 1, ... y 15). Las instrucciones no implementadas se han rellenado igualmente, apuntando a una rutina que procesa los opcodes desconocidos. En esta implementación se ha decidido ignorar estos, tratándose como si fueran un NOP:

```
.org 0x0FC0
; Tipo 0
.dw  ALG_EMU_CLRC
.dw  ALG_EMU_CLRV
...
```

De esta forma, la tabla que lista para realizar una indexación: Si accedemos a la dirección 0x0FC0 más el campo “opcode” de la instrucción más 16 veces el valor de su campo “tipo”, se obtiene la dirección de la rutina que buscamos. Esto exactamente es lo que se implementa en esta parte de la rutina “ALG_EJECUTA_1_INST”.

Unos ejemplos de rutinas emuladoras de instrucciones son estas, para las instrucciones OUT y LD, respectivamente:

```
; OUT ----------------
ALG_EMU_OUT:
      ; Asegurar modo superior
      rcall TEST_MODO_SUP
      brcs  EMU_SHLA_NO_OVR         ; Un RET cercano

      ; Hay que colocar en temp el dato, y en temp2 el puerto:

      ; Puerto en CRX
      rcall ALG_PON_Z_A_REGISTRO_CRX
      adiw  ZL,1                    ; Byte bajo
      ld    temp2,Z
```

```
; dato en CR
rcall ALG_PON_Z_A_REGISTRO_CR
adiw  ZL,1                    ; Byte bajo
ld    temp,Z

; Realizar el OUT en si:
rjmp  ALG_OUT_PORT            ; El RET se realiza alli



; LD -----------
ALG_EMU_LD:
      rcall ALG_OP_EN_X_A_X   ; X=MP[X]

      rcall ALG_PON_Z_A_REGISTRO_CR

      st    Z+,XH
      st    Z+,XL

      ret
```

Como se ve el código esta muy optimizado en espacio y se han usado el mayor número posible de subrutinas, debido al escaso espacio de programa del microcontrolador, lo que puede dificultar un poco la claridad de los fuentes del firmware.

- Etapa post-ejecución. Tras la ejecución de la instrucción en sí, se revisan las fuentes de interrupción por modo rastreo o por periféricos, saltando al vector correspondiente en cada caso. Por último se comprueba si se ha llegado a alguno de los *breakpoints*, en caso de estar estos definidos, parando la ejecución.

4.3.5.8. Subsistema de comunicaciones

Como se vio en el punto 4.3.5.4. la entrada del puerto COM1 (el usado para las comunicaciones con el PC y la aplicación "Comunicador") es manejada por la UART del microcontrolador y se usan su interrupción de byte recibido para la entrada de datos.

La rutina que se procesa esta interrupción es la que se encarga de formar la trama recibida byte a byte, en un buffer en la RAM ("RX_BUFFER"). También se usa una variable que indica el número de byte recibido dentro de la trama. En cada interrupción se coge el nuevo byte, se añade a la trama y se devuelve el control al programa principal.
En caso de detectar un error como un error en el byte de cabecera, o un campo de longitud mayor del máximo permitido (44 bytes) se desecha la parte de trama que se llevase construida y se vuelve a esperar un comienzo de trama válido. Las tramas desde el PC deben comenzar por 0x45, lo que facilita la sincronización de trama.

Es en esta rutina de proceso de interrupción también donde se implementa el nivel 2 del protocolo de comunicaciones, la capa de enlace:

- Con la llegada de cada byte se reinicia el temporizador hardware del microcontrolador TIMER1, con un tiempo de 4.4ms. Si el temporizador termina, genera una interrupción que provoca que se deseche la trama actual y se vuelva a esperar el comienzo de una nueva.
- Cuando se ha recibido la trama completa, se realizan una serie de pruebas:
    - Si la trama es una trama NACK, se reenvía la última trama enviada (guardada en RAM en el buffer "TX_BUFFER") de nuevo al PC.
    - Se calcula el XOR de todos los bytes de la trama, incluyendo el LRC, lo que debería producir un resultado de cero. Si no es así, se envía una trama de NACK al PC solicitando el reenvío de la trama, al haberse producido al menos un error en la transmisión.
    - Si todo es correcto, se notifica a la siguiente capa del protocolo, el procesador de comandos. Esto se realiza mediando la señalización en una variable en RAM ("CMD_A_EJECUTAR") el primer byte de datos de la trama (recordando el formato mostrado en 4.3.4.3, se ve que corresponde

con el código de comando a ejecutar). Esta variable normalmente esta a cero indicando que no hay una trama lista para procesar. Además, se inhiben más interrupciones de la UART, de forma que no se pueda sobrescribir el buffer de recepción, ya que por razones de limitación de RAM sólo existe uno.

En cuanto a la implementación de la llamada capa 3, o de comandos, se ha realizado en la rutina “PROC_CMD”.
Esta rutina no es llamada directamente por la rutina que procesa las interrupciones, lo que podría pensarse como mejor solución por el siguiente motivo: No se tiene control sobre en que momentos puede iniciarse la ejecución de un comando. Algunos de estos comandos implican accesos a los bancos de memoria externa, lo que representa un riesgo al poder interrumpir una transacción de bus I2C que estuviese ejecutando el programa principal. Esto provocaría la perdida irremediable de datos que no se puede permitir. Una solución podría ser inhibir las interrupciones durante transacciones I2C, pero esto conllevaría un retraso variable en el proceso de interrupciones como las de la UART que no los permiten.

Por todo esto, la rutina de procesado de comandos se llama desde el programa principal, desde los bucles centrales de las rutinas de:

- Menú principal.
- Ver pantalla de Algorítmez.
- En estado de ejecución continua.

Esta rutina se ha implementado como una búsqueda en una tabla de pares comando – dirección de rutina, parecido al método usado para la ejecución de instrucciones Algorítmez. Si el número de comando no esta implementado, se devuelve un campo de datos de nivel 3 solamente con un byte, el código de respuesta normal, igual al número de comando con el bit superior a 1.
Tras procesar el comando, esta rutina devolverá una trama de respuesta y volverá a activar las interrupciones de la UART para que nuevos comandos puedan ser recibidos.

4.3.6. Ensamblador independiente

Como ya se vio, las clases Java del simulador de Algorítmez se diseñaron de forma que fueran fácilmente reutilizables. Aprovechando esto, se ha creado una pequeña aplicación en modo línea de usuario: "AlgEns.exe"

Aunque esta aplicación se ha creado en lenguaje Java, se ha usado la herramienta "jexegen" de Microsoft para convertirla en un ejecutable binario para plataforma Windows, de forma que se facilite su uso independiente.

A esta aplicación se le pasa un parámetro en la línea de comandos, indicando un fichero que contiene un programa fuente de ensamblador Algorítmez. El programa lo carga y llama al método "Ensamblar" que ya se ha visto de la clase CVMAlgoritmez, mostrando por la salida estándar los mensajes y errores que se generan.

También se crea un fichero objeto en el mismo directorio donde se encuentra el fuente, con el mismo nombre de fichero que este pero con extensión ".obj":

```
C:\Archivos de programa\Comunicador Algorítmez>algens ejemplos/RCI.alg
AlgEns - Ensamblador de Algoritmez
   P.F.C. Jose Luis Blanco Claraco @ 2001-2002
------------------------------------------------
Emsamblando fichero: ejemplos/RCI.alg
Ensamblando... pasada numero 1
Ensamblando... pasada numero 2
Ensamblado finalizado SIN errores
     Se han encontrado 4 simbolos
     Se han encontrado 24 etiquetas
Se ha creado fichero objeto: ejemplos/RCI.obj

C:\Archivos de programa\Comunicador Algorítmez>_
```

Solamente se ha modificado la clase CVMAlgoritmez para que la tabla de periféricos que se crea en memoria automáticamente sea la que se muestra en 4.3.2.3, es decir, para reflejar los dos nuevos periféricos del entrenador que pueden generar una interrupción.

El motivo de crear esta aplicación es para facilitar al programa "Comunicador" el ensamblado de programas Algorítmez, reutilizando el código ya escrito en Java.

4.3.7. Software del comunicador

4.3.7.1. Introducción

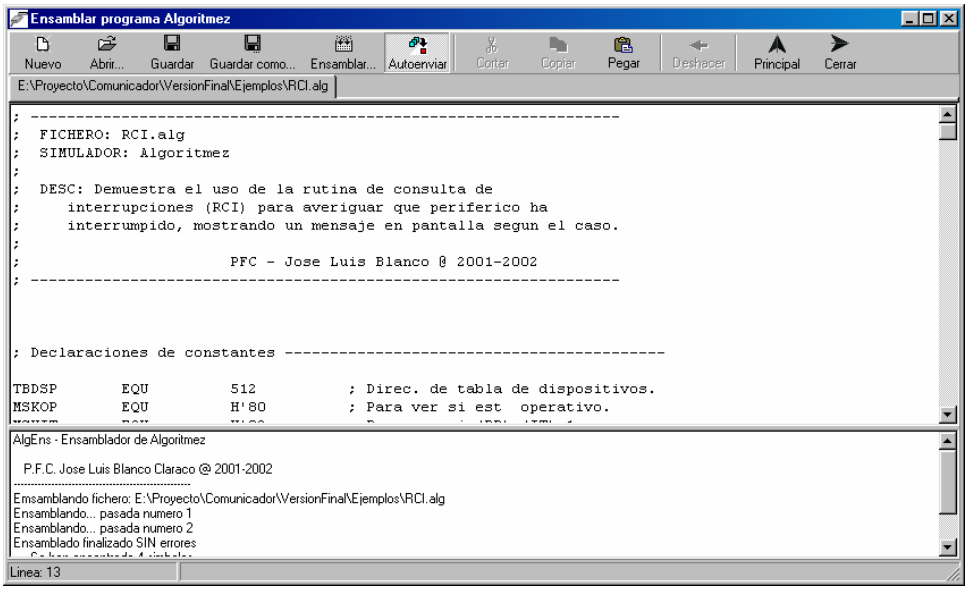
En la descripción del entrenador hardware de Algorítmez se ha visto como este es capaz de emular un programa almacenado en un banco de memoria, y como se ha definido un protocolo de comunicaciones que permite, entre otras cosas, grabar este programa en el entrenador. Por lo tanto es imprescindible el uso de un programa en el PC que implemente al menos esta funcionalidad.

Para cumplir con esta función se ha desarrollado la aplicación “Comunicador”, que se ejecuta sobre plataforma Windows y se comunica con el entrenador vía puerto serie.

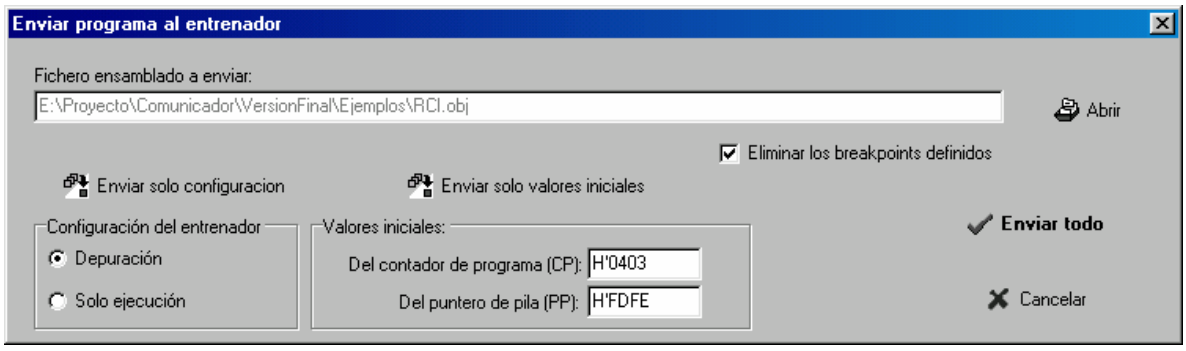
En este apartado solamente se verán algunos puntos importantes de esta aplicación. En el anexo A5.4 se detallan todas las capacidades de esta aplicación y una breve explicación de como usarla.

4.3.7.2. Ensamblado y envío de programas

Para el ensamblado de programas fuente Algorítmez se usa el programa AlgEns visto en el apartado anterior, pasándole como argumento la ruta completa del fichero fuente en edición. Luego se pide un *pipe* al sistema operativo, para unir la salida estándar del proceso AlgEns con el programa Comunicador, de forma que se puedan mostrar los mensajes del ensamblador en la misma ventana del editor:

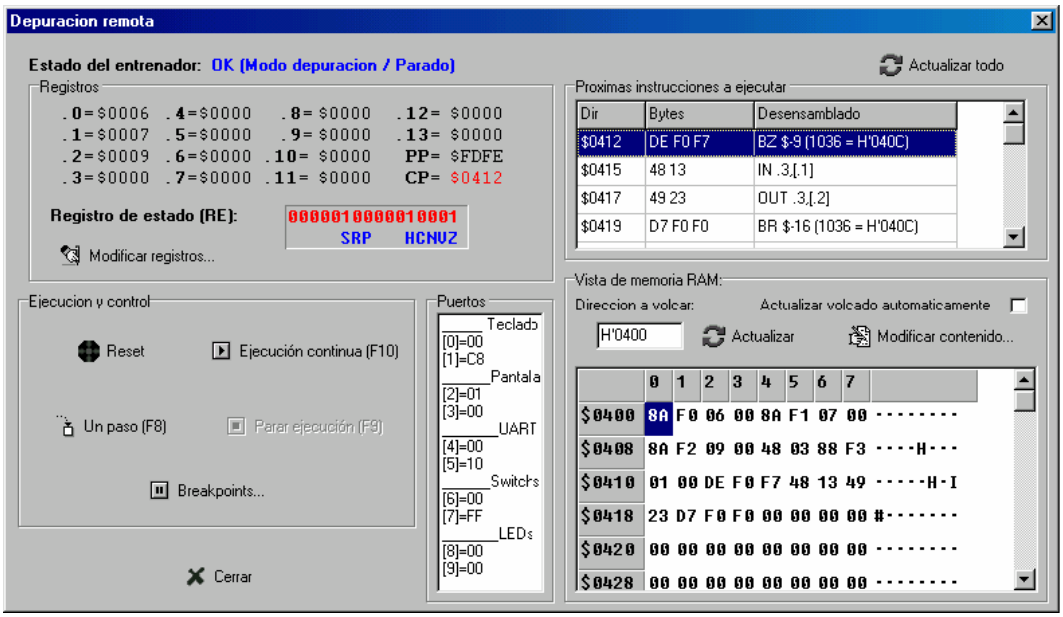


Si se comprueba que no se han producido errores, se puede enviar al entrenador el contenido del programa ensamblado que habrá quedado guardado en un fichero objeto, con el formato mostrado en 4.3.8. El módulo de envío de programas permite además configurar el entrenador en sus modos de depuración o de sólo ejecución:



4.3.7.3. Depuración remota

Desde el módulo de depuración remota se muestra el estado de la máquina virtual Algorítmez emulada en el entrenador. Esto se consigue usando los comandos descritos en el apartado 4.3.4.3:



4.3.8. Ficheros objeto

El formato de los ficheros objeto creados por el programa AlgEns es el siguiente:

Campo	Offset (bytes)	Longitud (bytes)	Descripción
Magic	0x00000	0x00004	Es constante = 0x44776172 Asegura que es un fichero del formato adecuado.
Contenido	0x00004	0x10000	La imagen de los 64kbytes de memoria de Algorítmez con el programa ensamblado.
Valor inicial de CP	0x10004	0x00002	Depende del programa.
Valor inicial de PP	0x10006	0x00006	Siempre a 0xFDE
TAMAÑO TOTAL		**0x10008**	

TABLA 18. Formato de los ficheros objeto

Donde se puede ver que contiene todos los datos necesarios para configurar el entrenador hardware para la ejecución de un programa concreto.

Además, el ensamblador introduce una tabla de periféricos en la dirección 512 de memoria, aunque el programa ensamblado puede sobrescribir dicha zona sin problema. Esta tabla de periféricos se describió en el punto 4.3.2.3.

4.3.9. Estudio de rendimiento

Se ha realizado una prueba ejecutando un programa Algorítmez que contiene un bucle donde se descuenta desde $1100 hasta $0000. Se ha emulado tanto en el simulador software como en el entrenador hardware, obteniéndose en este ultimo un tiempo de ejecución de 7.5 segs. Observando en el simulador software el número de instrucciones, se ve que el bucle ejecuta 8703 instrucciones, de forma que tenemos un valor aproximado para el entrenador Algorítmez de:

1160 instrucciones / segundo

Este valor puede verse ligeramente aumentado o disminuido dependiendo de la distribución de tipo de instrucciones que contenga cada programa, así como del uso que se haga de los periféricos, ya que usar tanto la pantalla como la UART simulada conllevan un coste adicional en procesamiento por parte del firmware.

5. Líneas de futuro

Las aplicaciones que se han desarrollado en este proyecto son independientes entre sí. Incluso se han definido rigurosamente el interfaz y el protocolo de comunicaciones usado por el “Comunicador” y el sistema entrenador, haciendo de esta forma posible la ampliación, mejora o especialización de cualquiera de las aplicaciones por separado.

El sistema entrenador es la parte más susceptible a líneas de trabajos futuros:

- Ampliación del hardware: Los dos puertos de entrada y salida generales añadidos a la máquina Algorítmez, podrían ser conectados a un bus de expansión de forma que se puedan montar prototipos con conversores analógicos digitales, analógicos digitales, sondas de telemedida, interfaz a un disco duro, etc...
- Ampliación de software: Sobre la máquina virtual Algorítmez se puede implementar cualquier aplicación, o incluso si se realiza una interfaz a un disco duro externo (o emulador del mismo), el sistema operativo “Mono Algorítmez” descrito en [1-Gregorio Fernández].
- Diseño e implementación de otros prototipos basados en este diseño, e incorporando controladores de otros periféricos o de acceso directo a memoria (controladores DMA).
- Hay que señalar además que el microcontrolador que se ha usado es perfectamente compatible con otros modelos superiores de la familia AVR, por lo que se podría usar el mismo firmware con un microcontrolador de gama alta, lo que proporcionaría mayor velocidad de emulación además de permitir un mayor número de puertos de entrada y salida para nuevos periféricos.

Así mismo el programa ensamblador en línea de comando para Algorítmez (AlgEns.exe) podría ser ampliado para soportar múltiples módulos en ensamblador, declaraciones externas, etc... lo que facilitaría el diseño de programas de una considerable envergadura para el entrenador.

6. Conclusiones

Se han desarrollado las herramientas que propusieron para la docencia de prácticas de la asignatura “Fundamentos de Computadores”, consistentes en los dos simuladores software de Símplez y Algorítmez, haciéndolos accesibles a todos los alumnos fácilmente vía web.

Además, se ha desarrollado un sistema emulador hardware con aplicaciones no solo en el campo de la docencia, sino incluso en prácticos casos reales, al comportarse como un verdadero sistema microprocesador de propósito general.

6. Bibliografía de referencia

[1] Bibliografía básica con las especificaciones de los dos microprocesadores Símplez y Algorítmez:
“Conceptos básicos de arquitectura de computadores y sistemas operativos” 1998 (Tercera edición) Gregorio Fernández.

[2] Algoritmos y técnicas para programas ensambladores:
“Software de sistemas” 1988 Leland L. Beck

[3] Documentación de JDK, Sun Microsystem.
http://java.sun.com

[4] Datasheet del microcontrolador AT90S8515:
http://www.atmel.com/atmel/acrobat/doc0841.pdf

[5] Datasheet de memoria eeprom I2C, AT24C512:
http://www.atmel.com/atmel/acrobat/doc1116.pdf

[6] Documento sobre displays LCDs:
“How to control a HD44780-based Character-LCD”
http://home.iae.nl/users/pouweha/lcd/lcd.shtml

[7] Web sobre interfaz con teclados PS/2 de PC:
“Interfacing the PC”
http://www.beyondlogic.org/

ANEXOS

A1. Requisitos del sistema

- Para la visualización de los simuladores desde la web, se requiere:

  - Navegador Internet Explorer 5.0 o superior. (Recomendado)
  - Netscape Communicator 4.3 o superior.

Ambos con soporte de Java habilitado.

- Para la ejecución de los simuladores en su versión ejecutable (.exe) para Windows, solo se requiere Windows 95 o superior.

- Para la ejecución de los simuladores en su versión JAR, se requiere tener instalado el JDK 1.1 o JRE 1.1 o superior, e incluir en el PATH la ruta hasta los programas "java" y "javaw" (Este último solo en Windows). Esta versión funciona tanto en sistemas Windows como Linux RedHat 7 o derivados y superiores.

- El programa comunicador para el entrenador de Algorítmez, funciona sobre Windows 95 o superior, y requiere para su instalación unos 5 Mbs de espacio libre en disco duro, un puerto serie libre y pantalla con profundidad de color de 16 bits o superior.

Todas las aplicaciones están diseñadas para una resolución mínima de pantalla de 800x600 píxels, aunque se recomienda la de 1024x768.

A2. Componentes Java

Para los simuladores en Java de Símplez y Algorítmez, se han creado unos componentes comunes, todos ellos en un *package* llamado "componentes". La lista de estos componentes, algunos visuales y otros no, se detalla a continuación:

DESCRIPCIÓN	
Nombre de la clase:	CBorde
Descendiente de:	Component
Descripción:	Componente visual que dibuja un rectangulo con efecto de relieve en toda su superficie.

MÉTODOS ÚTILES

CBorde()	Constructor. Sin parámetros.

DESCRIPCIÓN	
Nombre de la clase:	CTimer
Descendiente de:	Thread

Descripción:	Componente no visual que permite la ejecución a períodos regulares de un método. La longitud del intervalo se puede elegir al construir el objeto. Permite parar y reanudar posteriormente. El objeto que quiera recibir los avisos de este temporizador debe implementar la interfaz *Avisable*:
	
	`public interface Avisable {`
	`  public void OnTimer();`
	`}`
	
	Donde el método *OnTimer* es el que será llamado al cumplir cada intervalo de tiempo.

MÉTODOS ÚTILES

CTimer(long timer,Avisable obj)	Constructor. El intervalo del temporizador se pasa en *timer* en milisegundos. El objeto al que se avisa al cumplir cada intervalo, se pasa en *obj*. Al comienzo el temporizador esta inactivo.
void activa()	Activa el temporizador.
void desactiva()	Desactiva el temporizador.

DESCRIPCIÓN	
Nombre de la clase:	ConversionesUtiles
Descendiente de:	Object
Descripción:	Objeto no visual que permite multitud de operaciones matematicas y conversiones. Todos los métodos son estáticos, por lo que no es necesario instanciar ningún objeto para usar sus métodos.

MÉTODOS ÚTILES

String ArreglarRCs(String st)	Sustituye los retornos de carro por la secuencia retorno de carro + avance de línea.
String byte2Hex(byte x)	Pasa un byte a hexadecimal.
int byte2int_unsigned(byte b)	Pasa un valor de tipo *byte* a otra de tipo *int*, en el intervalo 0..255 en lugar de –128..127.
boolean CalcAcarreoEnBit_Resta(int a,int b,int nBit,boolean Carry,int nBits)	Devuelve **true** si se produciría acarreo al realizar la resta de *a* y *b*, en el bit número *nBit*. El acarreo inicial se pone a *Carry*. Esta función simula la operación de resta bit a bit para comprobar el acarreo solicitado. El valor *nBits* deberá ser de 8 o 16, según sea la resta de uno o dos bytes.
boolean CalcAcarreoEnBit_Suma(int a,int b,int nBit,boolean Carry)	Devuelve **trae** si se produciría acarreo al realizar la suma de *a* y *b*, en el bit número *nBit*. El acarreo inicial se pone a *Carry*. Esta función simula la operación de resta bit a bit para comprobar el acarreo solicitado.
Point CalculaPosicionDialogo(Dimension dimDlg)	Devuelve las coordenadas donde hay que colocar un cuadro de diálogo para que quede en el centro de la pantalla.
boolean CogeBit(int num,int nBit)	Devuelve el bit número *nBit* de la palabra *num*.
int dosBytes2int(byte bBajo,byte bAlto)	Construye un valor *int* con el valor de 2 bytes indicado.
String int16bits_signo2str(int x)	Pasa un *int* a cadena, usando como bit de signo el bit 15.
String int2Bin(int x,int nDigitos)	Pasa un *int* a binario, con el número de digitos dado.
String int2Dec(int x,int nDigitos)	Pasa un *int* a decimal, con el número de digitos dado.
String int2Hex(int x,int nDigitos)	Pasa un *int* a hexadecimal, con el número de digitos dado.
String int2oct(int x,int nDigitos)	Pasa un *int* a octal, con el número de digitos dado.

DESCRIPCIÓN	
Nombre de la clase:	CTabla
Descendiente de:	Component
Descripción:	Componente visual que muestra una tabla de numero de filas y columnas variables, con cabeceras de columna, selección de fila o celdas, menú contextual y obtención dinámica de los datos a visualizar.

MÉTODOS ÚTILES

public CTabla(OrigenDatosTabla odt)	Constructor. Todos los parámetros de configuración se obtienen del objeto *odt* que debe implementar la siguiente interfaz:
	
	`public interface OrigenDatosTabla {`
	`  public int nCols();`
	`  public int nFils();`
	
	`  public String elemento(int col,int fila);`
	`  public String tituloColumna(int col);`
	
	`  public int anchoColumna(int nCol);`
	`  public int filasPorPagina();`
	`}`

int getPrimeraLinea()	Devuelve la primera línea visible en la tabla.
void RecalculaEspacio()	Recalcula y redibuja la tabla. Util tras variar algun parámetro de numero de filas, columnas, etc...
void SeleccionarFilas(boolean f,int nColMin,int nColMax)	Si se quiere que se seleccionen las filas completas, pasar *f* a **true**. Los valores *nColMin* y *nColMax* indican el rango de columnas que se va a permitir seleccionar.
void setOdt(OrigenDatosTabla odt)	Permite cambiar el origen de datos de la tabla tras construirla.
void VeALinea(int nLin)	Mueve el scrollbar vertical para hacer visible la línea indicada.

DESCRIPCIÓN	
Nombre de la clase:	DlgPeticion
Descendiente de:	Dialog
Descripción:	Cuadro de dialogo que permite la entrada de un texto por el usuario. Contiene un texto indicativo de la petición y botones para aceptar y cancelar. Si se pulsa "Aceptar", al cerrar la ventana la propiedad *Ok* contendrá un **true**, y el texto de la respuesta estará en la propiedad *resultado.*

MÉTODOS ÚTILES

DlgPeticion(Frame frame, String title)	Constructor. Los parámetros son: El *Frame* padre de este dialogo, y un título a mostrar en el diálogo.
void PonDefecto(String lb,String ed)	Coloca el texto *lb* como texto de la petición, y *ed* como texto por defecto en la respuesta del usuario.

DESCRIPCIÓN	
Nombre de la clase:	DlgShowMessage
Descendiente de:	Dialog
Descripción:	Cuadro de dialogo que muestra un texto y un botón "Aceptar".

MÉTODOS ÚTILES

DlgShowMessage(Dialog parent, String title, boolean modal)	Constructor. Los parámetros son: El dialogo padre de este, y un título a mostrar en el diálogo.
void SetTexto(String txt)	Pone el texto indicado en el diálogo.

A3. Esquemas

En las siguientes páginas se incluyen, y por este orden, los siguientes esquemas:

- Esquema eléctrico del sistema entrenador de Algorítmez (Páginas 1/2 y 2/2)
- Imagen del fotolito usado para la insolación de la placa del prototipo para el entrenador.
- Imagen del fotolito superpuesta con los componentes, a forma de guía de montaje.

A4. Lista de materiales del entrenador

Nombre	**Cantidad**	**Descripción**
74HC04	1	
7805	1	REGULADOR DE VOLTAJE
AT24LC512	2	64Kb EEPROM I2C
AT90S8515-8PC	1	MICROCONTROLADOR
CABLE ALIMENTACIÓN 220V	1	
CABLE PLANO DE 10 CABLES	4	20 cms. aprox.
CAJA DE PLASTICO PARA ENTRENADOR	1	
CONDENSADOR 100nF	2	
CONDENSADOR 220nF	1	
CONDENSADOR CERÁMICO 22 pF	2	
CONDENSADOR ELECTROLÍTICO 100 μF/35V	1	
CONDENSADOR ELECTROLITICO 10μF / 35V	4	
CONECTOR CABLE PLANO 10 PINS (HEMBRA)	4	
CONECTOR CABLE PLANO 10 PINS (MACHO)	4	
CONECTOR DB-9 MACHO (CODADO) PARA C.I.	2	
CONECTOR PS-2 HEMBRA PARA C.I. (DIN 6)	1	
CRISTAL 7.3728 MHZ	1	
DIODO 1N4001	5	
FUSIBLE 0.02 A	1	
INTERRUPTOR ALIMENTACIÓN 220V	1	
LCD DE 2x16 CARACTERES COMPATIBLE CON HD44780	1	MODELO CEBEK C-2602
LED ROJO	9	
LED VERDE	2	
MAX232 o ST232	1	ADAPTADOR RS232-TTL
MICROSWITCH DE 8 INTERRUPTORES	1	
PLACA PARA INSOLAR POR UNA CARA.	1	TAMAÑO 100x200 mm
PORTAFUSIBLES	1	
PULSADOR (PARA RESET DE SISTEMA)	1	
RESISTENCIA AJUSTABLE 10KΩ (HORIZONTAL)	1	
RESISTENCIAS DE ¼ W (1 de 100KΩ, 2 DE 22KΩ ,11 DE 1KΩ)	14	
TRANSFORMADOR PARA C.I. 220V-12V AC (2 VA)	1	
ZOCALO 16 PINS (DIL)	2	
ZOCALO 40 PINS (DIL)	1	
ZOCALO 8 PINS (DIL)	2	

A5. Manuales de usuario

A5.1. Simulador de Símplez

A5.1.1. Instalación y ejecución

El programa simulador de Símplez se puede ejecutar de distintas formas:

1. Versión online. Basta con entrar en la página web del proyecto y abrir el enlace a “Simulador de Símplez”, “Versión online”. Si estamos usando el navegador Internet Explorer, aparecerá una ventana solicitando los derechos del applet:

Donde se debe responder que “Si” si se quiere acceder a programas guardados en ficheros en disco.

2. Versión ejecutable Windows. Hay que descargar el fichero *simSimplez.exe* de la web del proyecto. Este fichero no precisa de instalación especial, simplemente se ejecuta y se inicia el simulador de Símplez.
3. Versión JAR multiplataforma. Disponible en la web como fichero comprimido (ficheros *simSimplez.zip* y *simSimplez.tgz*). Una vez descargado este fichero, se coloca en una carpeta y se descomprime obteniendo estos tres ficheros:

- *SimSimplez.jar*: Donde están las clases de la aplicación comprimidas.
- *SimSimplez.bat*: Fichero script BAT para sistema Windows.
- *SimSimplez.sh*: Fichero script shell para sistema Linux.

Cada uno de los script ejecuta el programa *java* o *javaw* que debe ser accesible a través del *PATH*, para lo cual es necesario tener instalado el JDK1.1 o superior o el JRE1.1 o superior. Para el caso de plataforma Windows, si los ficheros JAR están asociados con el programa *javaw*, haciendo doble clic sobre el fichero JAR se abrirá automáticamente la aplicación sin necesidad de usar el fichero BAT.

Independientemente del método usado, aparecerá la ventana principal de la aplicación.

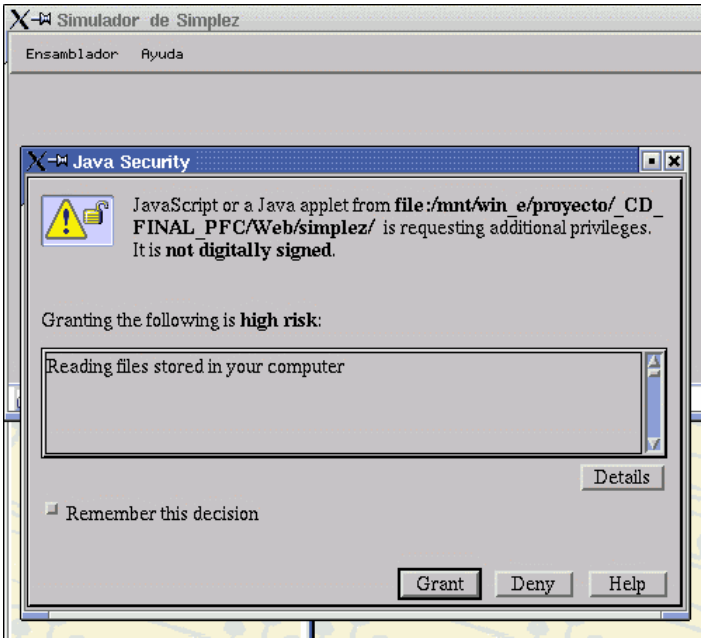
A5.1.2. Ventana principal

Esta ventana solo se proporciona como un método cómodo de acceder al menú de carga de programas, independiente de la ventana del navegador usado (para el caso de la versión en applet).

Contiene los siguientes menús:

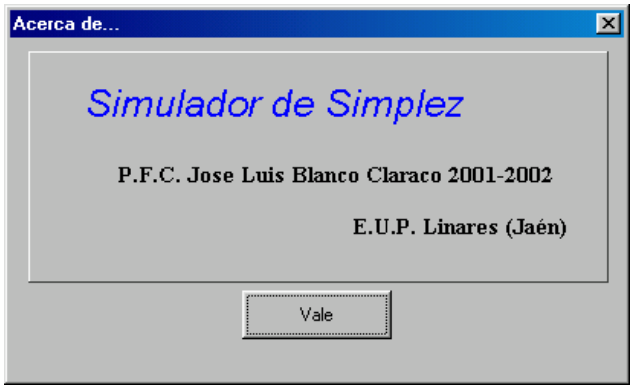
- Ensamblador
  - Nuevo programa: Abre una ventana de editor de programas en blanco.
  - Cargar programa: Permite cargar un programa guardado en un fichero en el editor de programas. En el caso de Netscape, aparecerán en este

momento dos ventanas solicitando los derechos de lectura y escritura de ficheros, que se deberá aceptar (*grant*):



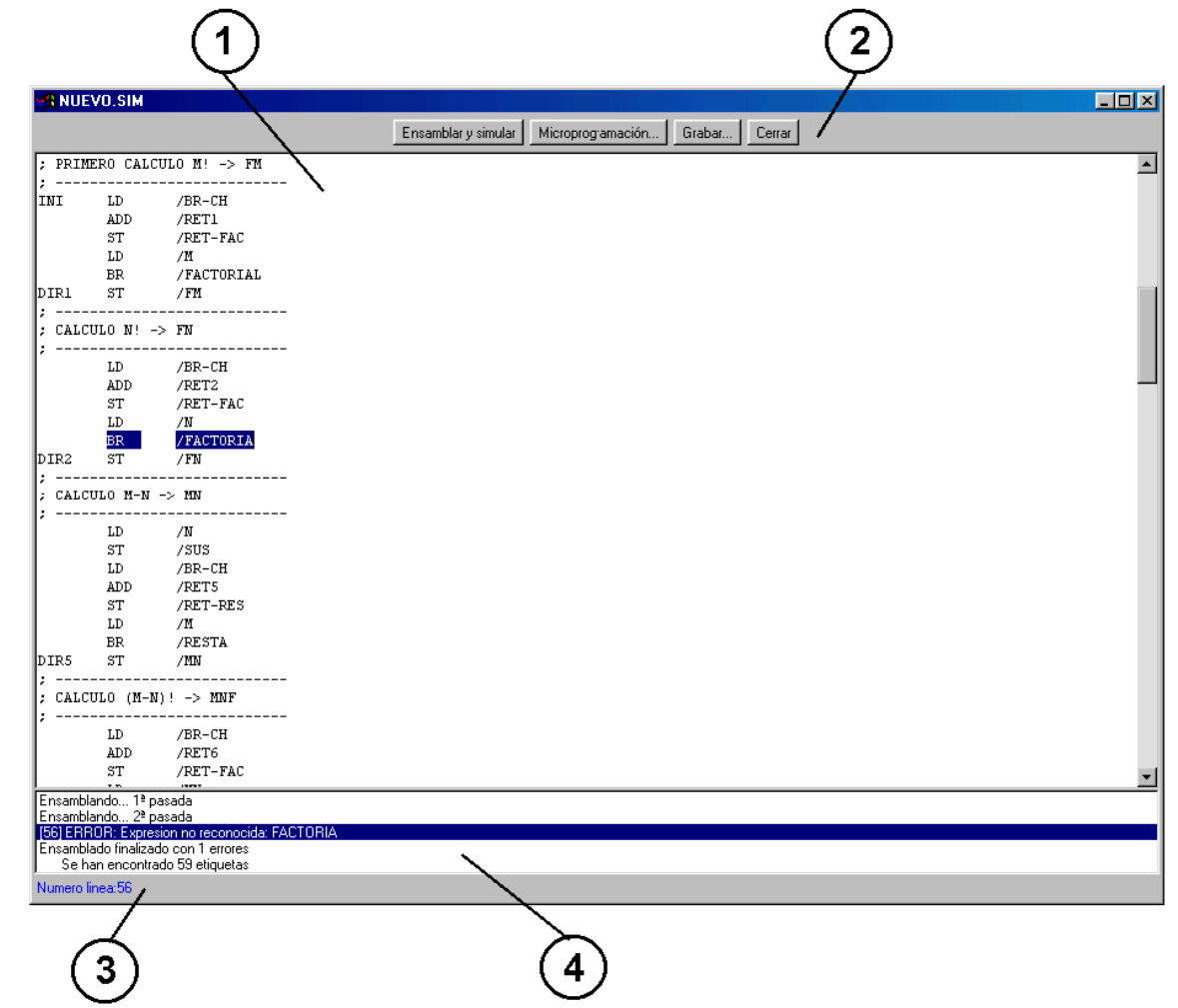
- Cerrar: Cierra la aplicación.

- Ayuda
  - Ayuda de Símplez: Solo disponible en la versión online. Abre una ventana del explorador con esta guía de usuario.
  - Acerca de: Muestra una ventana con los datos de la aplicación.



A5.1.3. Editor de programas

El editor de programas sirve de base tanto para editar los programas Símplez, editar la micro memoria de control para cada caso, y lanzar la simulación de los mismos.



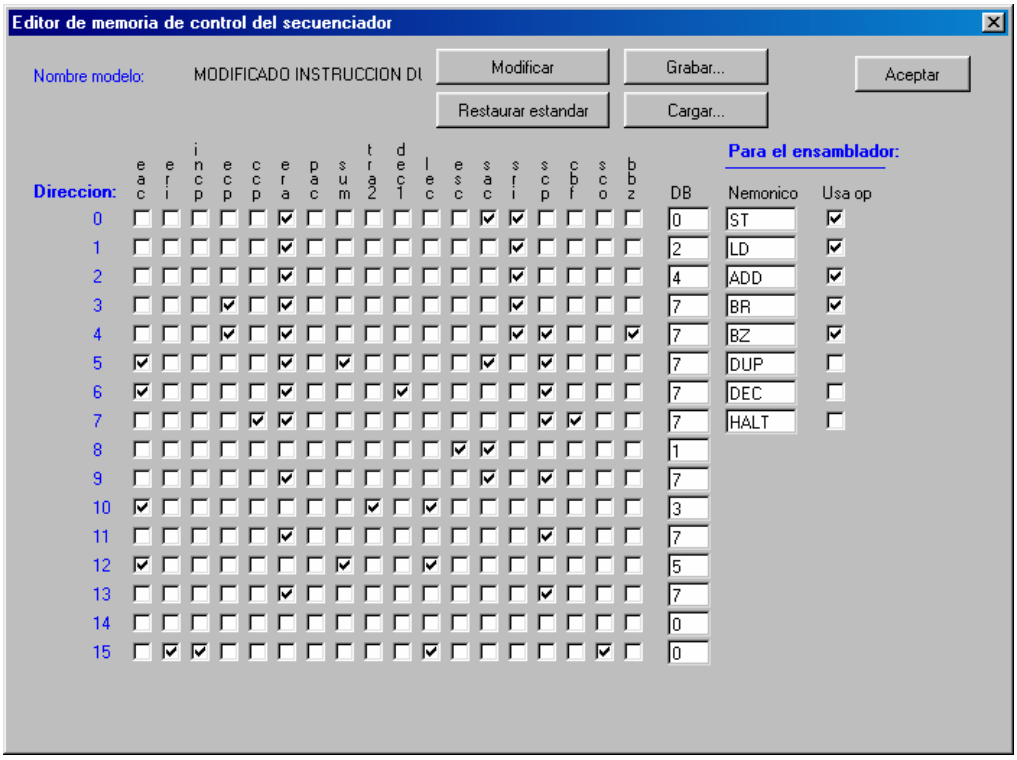
En esta ventana se pueden ver las siguientes partes:

1. Campo del editor, donde se puede ver y editar el programa.
2. Barra superior de botones. Los botones son los siguientes:
   - Ensamblar y simular: Ensambla el programa y muestra los mensajes del ensamblador en la zona de mensajes inferior. En caso de no producirse ningún error, se abrirá la ventana del simulador con el programa cargado en memoria y listo para iniciar la simulación.
   - Microprogramación: Permite ver y editar el contenido de la memoria de control del secuenciador de Símplez en el editor de micromemoria.
   - Grabar. Permite guardar el programa en un fichero. Es preciso haber aceptado los permisos solicitados si se ejecuta desde un navegador para que esta opción este disponible.
   - Salir: Cierra esta ventana del editor.

3. Barra inferior. Aquí se muestra el número de línea sobre la que se encuentra el cursor en cada momento.
4. Zona de mensajes, donde se muestran los mensajes y los errores generados por el ensamblador. Haciendo doble-clic sobe un error, se resaltará automáticamente la línea correspondiente al error en el programa. Esta lista se puede esconder si se desea pulsando el botón derecho sobre el campo de texto del editor, y seleccionando en el menú contextual la opción “Mostrar/ocultar mensajes del ensamblador”.

A5.1.4. Editor de micromemoria

Este editor se abre desde el editor de programas Símplez, y permite modificar las características del secuenciador de Símplez. Estos cambios solo se aplican a la sesión abierta del editor de programas, y las simulaciones lanzadas desde este, de forma que es necesario cargar el micromodelo cada vez que se carga el programa en ensamblador, en el caso de usar una micromemoria distinta a la estándar.



Las partes y botones de este editor son las siguientes:

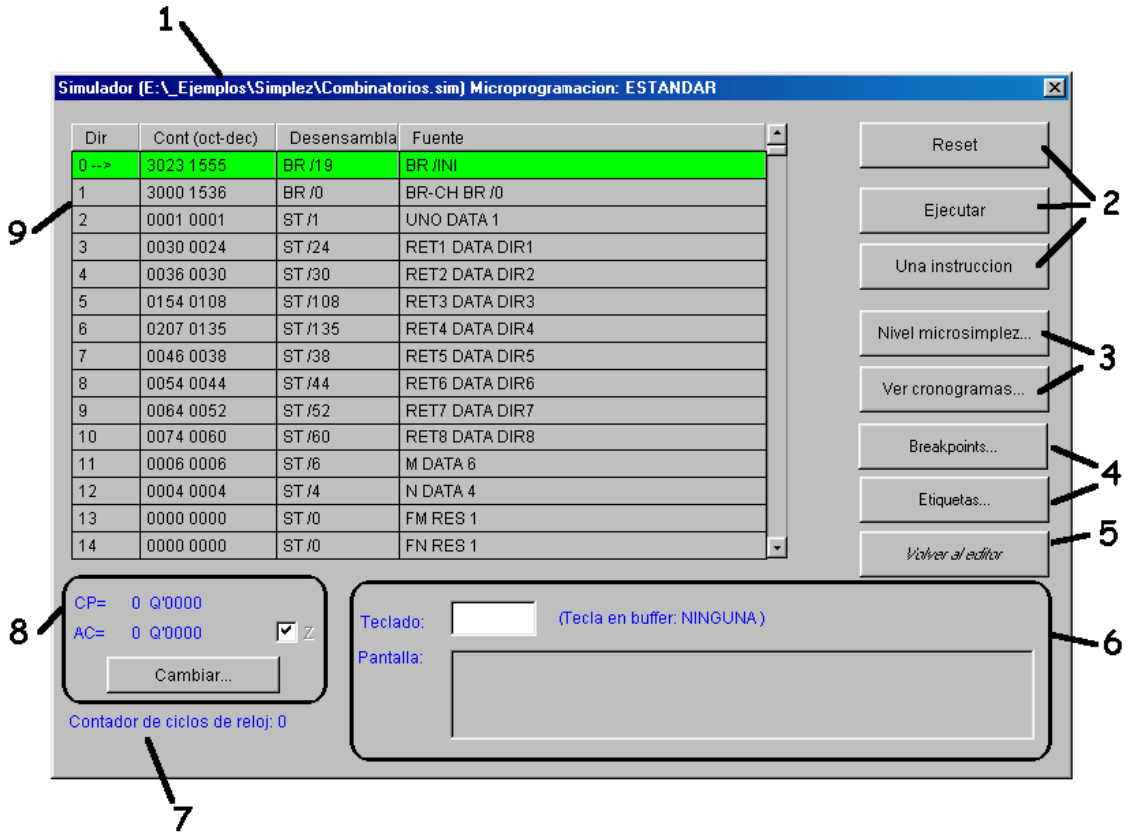
- Nombre de modelo: “Estándar” en el caso del contenido e instrucciones estándar de Símplez, o el nombre que el usuario quiera dar si ha realizado una modificación a la misma.
- Boton “Modificar”: Pide un nuevo nombre para el modelo y habilita la edición de las casillas de las señales, deshabilitada por defecto.
- Botones “Grabar” y “Cargar”: Permiten guardar y cargar todos los datos del micromodelo en un fichero, con extensión por defecto de “.mic”
- Matriz de memoria: Esta formada por las 16 filas de 18 casillas y sus correspondientes campos DB. Cada fila representa una palabra de 21 bits en

la memoria de control del secuenciador. Para el significado de estas palabras, ver la bibliografía [1-Gregorio Fernández].

- Datos para el ensamblador: Permite introducir los nemónicos de las instrucciones con el opcode correspondiente al número de fila (desde 0 hasta 7), así como especificar si la instrucción usa o no el campo de operando. Estos datos facilitan la implementación y uso de nuevas instrucciones.

A5.1.5. Ventana del simulador

Tras ensamblar un programa Símplez, si el código no contiene errores, se abre automáticamente la ventana de simulación:



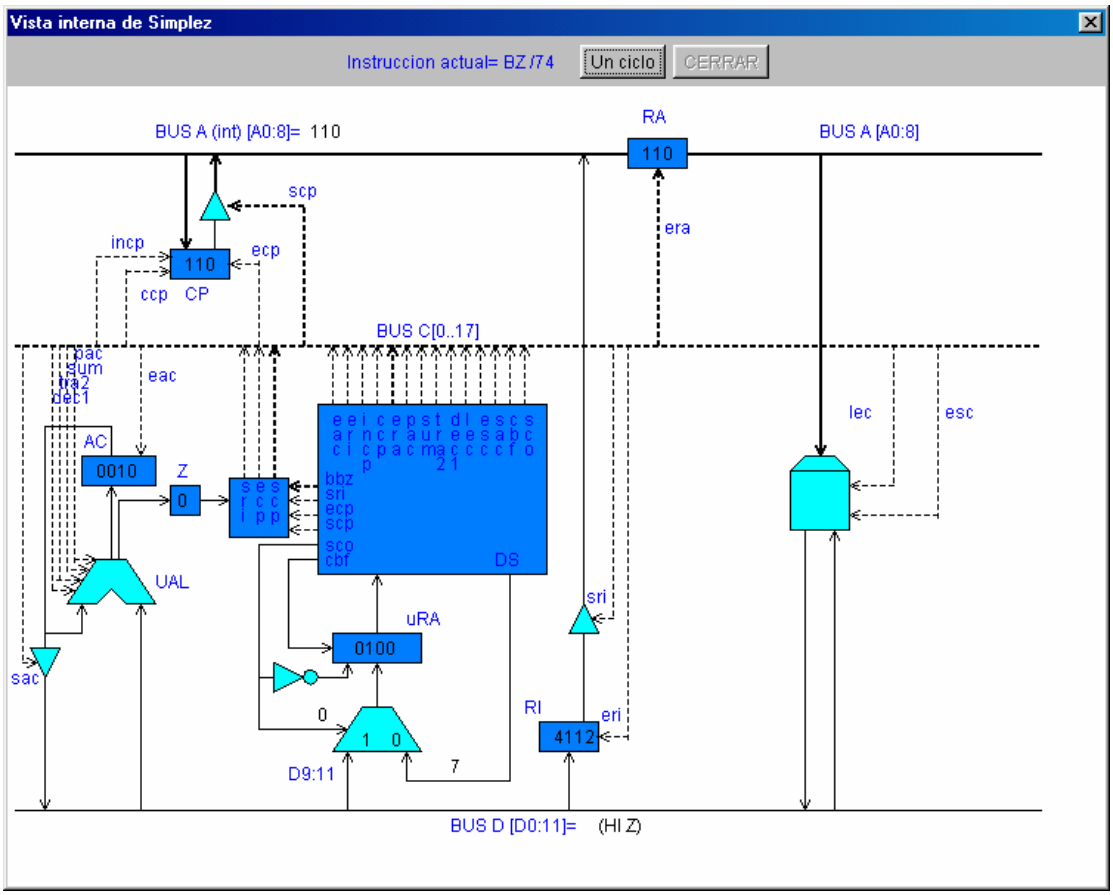
Donde los elementos más importantes son:

1. Título: Donde se puede ver el nombre tanto del fichero fuente ensamblador que se está simulando, como el nombre del micromodelo usado.
2. Botones de control principales:

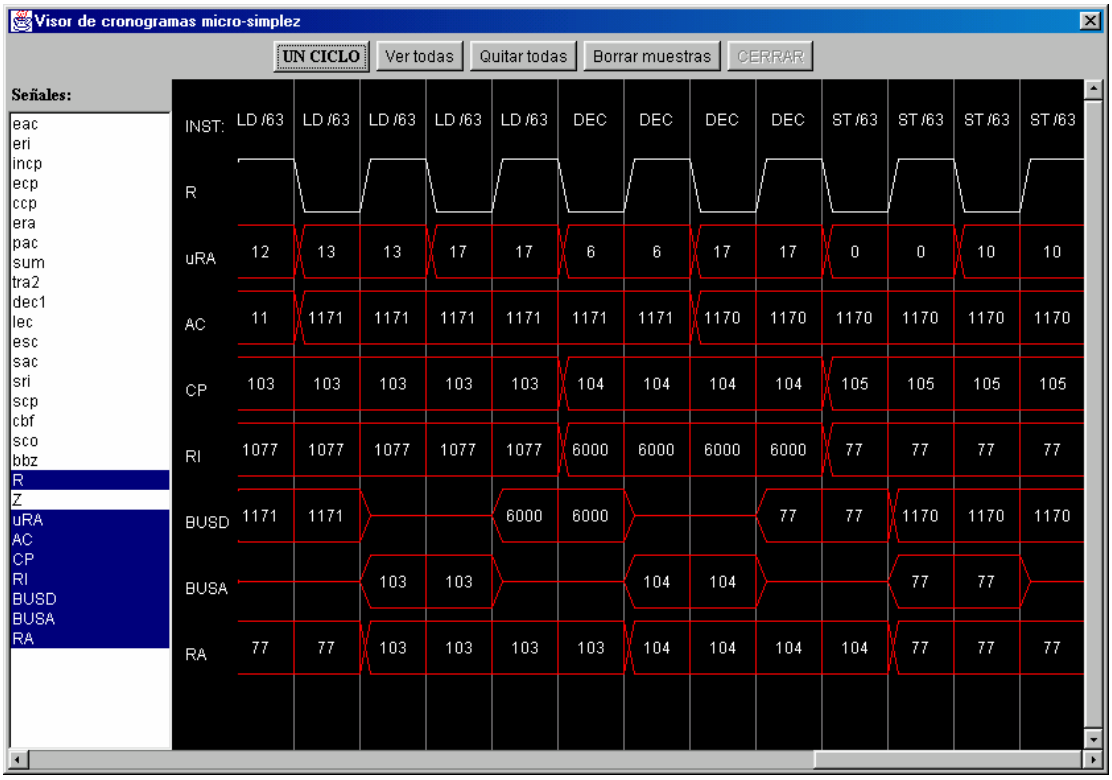
- Reset: Llama al método “Reset” de la máquina virtual Símplez, reiniciando el estado y recargando el programa ensamblado en memoria.
- Una instrucción: Llama al método auxiliar que vimos en el apartado anterior, que se encarga de dar pulsos a la señal de reloj de la máquina virtual hasta que se termine la ejecución de una instrucción completa.
- Ejecutar: Este botón inicia el modo de ejecución continua: Se ejecutan instrucciones continuamente hasta llegar a un HALT o a un breakpoint. Para parar la ejecución continua manualmente, se puede pulsar el botón “Parar”, que sólo aparece cuando se está en este modo.

3. Modos de ejecución secundarios:

- Modelo microsímplez: En este modo se puede ejecutar un programa a nivel de ciclos de reloj, mostrándose en pantalla un esquema de la arquitectura interna del microprocesador y el estado de todas las señales:

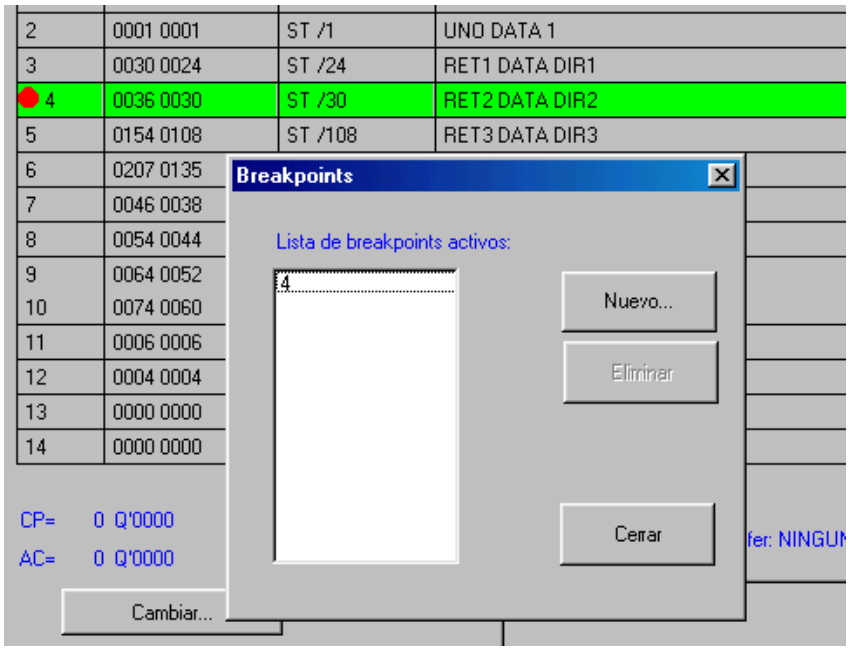


- Modo cronogramas: También se puede ejecutar a nivel de ciclos de reloj, viéndose en este caso representados en forma de cronogramas los estados de todas las señales, registros y buses del sistema:

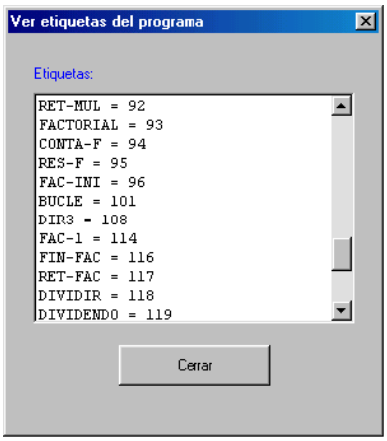


De todas las señales disponibles en la lista de la izquierda, sólo se muestran en cada momento las que estén marcadas, para mayor claridad de los cronogramas. Independientemente de que una señal este o no marcada, se guardan todas las muestras, pudiendo visualizarse posteriormente.

4. Otros botones:
   - Breakpoints (Puntos de ruptura): Muestra una ventana donde se ven los breakpoints activos, permitiendo quitarlos o añadir nuevos:



   - Etiquetas: Muestra una lista con las etiquetas encontradas en el programa y sus direcciones correspondientes:



5. Volver al editor: Cierra esta ventana de simulador para volver al editor de programas en ensamblador. Al salir se elimina totalmente el estado de la máquina virtual, incluyendo memoria, puntos de ruptura, etc...

6. Consola de Símplez: Estos controles representan los dos dispositivos de E/S de Símplez: La pantalla y el teclado. Como se ve, se muestra además el carácter que está preparado por el teclado. Si se pulsa una tecla estando ya el buffer ocupado, la nueva tecla se pierde.

7. Contador de ciclos: Se muestra el número de ciclos de reloj que han transcurrido desde el reset de la máquina virtual. Este contador refleja los ciclos ejecutados independientemente del modulo desde el que se hayan ejecutado (Micromodelo, cronogramas o ventana principal del simulador).

8. Registros: Aquí se puede ver el contenido del acumulador (AC), del contador de programa (CP) y del biestable Z. Además, se permite modificarlos manualmente en cualquier momento.

9. Tabla de memoria: Se muestran 512 filas, una por cada posición del espacio de direcciones de Símplez. Las columnas de izquierda a derecha, representan:

    - Dirección.
    - Contenido de esa palabra de memoria (o de E/S) tanto en decimal como en octal.
    - Desensamblado: Interpretación de esa palabra como una instrucción Símplez, siguiendo el micromodelo seleccionado.
    - Código fuente: La línea del programa original que está relacionada con esa dirección.

A5.1.6. Referencia rápida de Símplez

Ensamblador

- Directivas:
  - ORG <Dirección>: Modifica la dirección donde el programa ensamblador almacenará las instrucciones o datos que sigan a continuación. Si se omite, la dirección por defecto es cero.

```
Ejemplo:
        ORG          H'100
```

  - END: No tiene argumentos. Indica el final de un programa. Su uso es obligatorio.

- Pseudo instrucciones:
  - DATA <lista de elementos>: Genera una lista de palabras con los valores indicados, separados por comas. Se pueden usar cadenas con comillas dobles, y usar prefijos para valores en binario (B'), octal (Q'), decimal (D') y hexadecimal (H').

```
Ejemplo:
        DATA    "Mensaje1", H'03, B'10010001
```

  - RES <Numero>: Reserva un numero de palabras en memoria.

```
Ejemplo:
        RESULT  RES  2
        CONTA   RES  1
```

Instrucciones

Estas son las instrucciones por defecto de Símplez:

Opcode	Nemónico	Descripción
0	ST	Guarda el contenido del acumulador en la dirección indicada en el operando.
1	LD	Carga en el acumulador el contenido de la dirección indicada en el operando.
2	ADD	Suma al acumulador, el contenido de la dirección indicada por el operando.
3	BR	Salta a la dirección indicada por el operando, incondicionalmente
4	BZ	Salta a la dirección indicada por el operando, solo si el indicador Z esta activado. Sino, sigue ejecutando normalmente
5	CLR	Carga un cero en el acumulador
6	DEC	Decrementa en una unidad el acumulador
7	HALT	Paraliza el estado de la máquina

Periféricos

Los dos periféricos, pantalla y teclado, tienen asignados cada uno dos puertos de E/S, mapeados en direcciones de memoria:

Periférico	Dirección del puerto	Descripción
Pantalla	508 = Q'774	Estado. Bit bajo a 1 cuando esta preparada.
	509 = Q'775	Datos. Salida de carácter a imprimir.
Teclado	510 = Q'776	Estado. Bit bajo a 1 cuando hay un carácter preparado.
	511 = Q'777	Datos. Entrada de carácter recibido

A5.2. Simulador de Algorítmez

A5.2.1. Instalación y ejecución

El programa simulador de Algorítmez se puede ejecutar de distintas formas:

4. Versión online. Entrando en la página web del proyecto y abrir el enlace a "Simulador de Algorítmez", "Versión online". Si estamos usando el navegador Internet Explorer, aparecerá una ventana solicitando los derechos del applet:

Donde se debe responder que "Si" si se quiere acceder a programas guardados en ficheros en disco.

5. Versión ejecutable Windows. Hay que descargar el fichero *simAlgoritmez.exe* de la web del proyecto. Este fichero no precisa de instalación especial, simplemente se ejecuta y se inicia el simulador.

   Versión JAR multiplataforma. Disponible en la web como fichero comprimido (ficheros *simAlgoritmez.zip* y *simAlgoritmez.tgz*). Una vez descargado este fichero, se coloca en una carpeta y se descomprime, con lo que se tienen estos tres ficheros:

   - *SimAlgoritmez.jar*: Donde están las clases de la aplicación comprimidas.

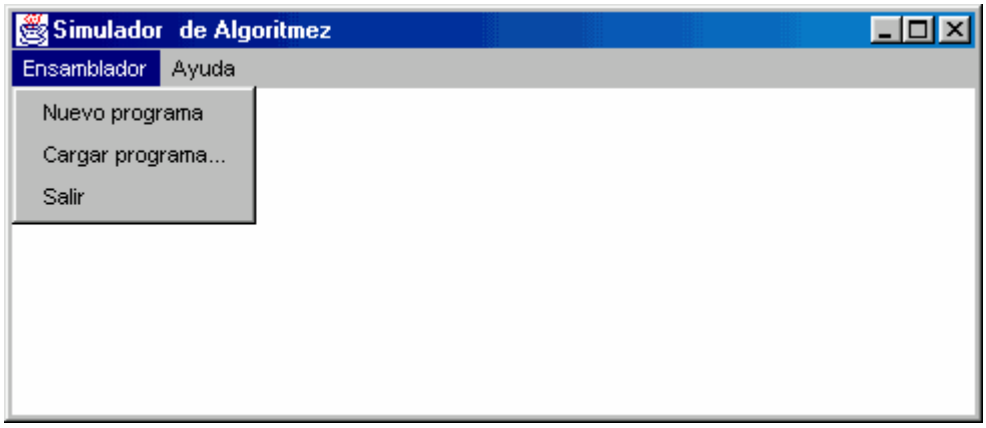
- *SimSimplez.bat*: Fichero script BAT para sistema Windows.
- *SimSimplez.sh*: Fichero script shell para sistema Linux.

Cada uno de los script ejecuta el programa *java* o *javaw* que debe ser accesible a través del *PATH*, para lo cual es necesario tener instalado el JDK1.1 o superior o el JRE1.1 o superior. Para el caso de plataforma Windows, si los ficheros JAR están asociados con el programa *javaw*, haciendo doble clic sobre el fichero JAR se abrirá automáticamente la aplicación sin necesidad de usar el fichero BAT.

Independientemente del método usado, aparecerá la ventana principal de la aplicación.

A5.2.2. Ventana principal

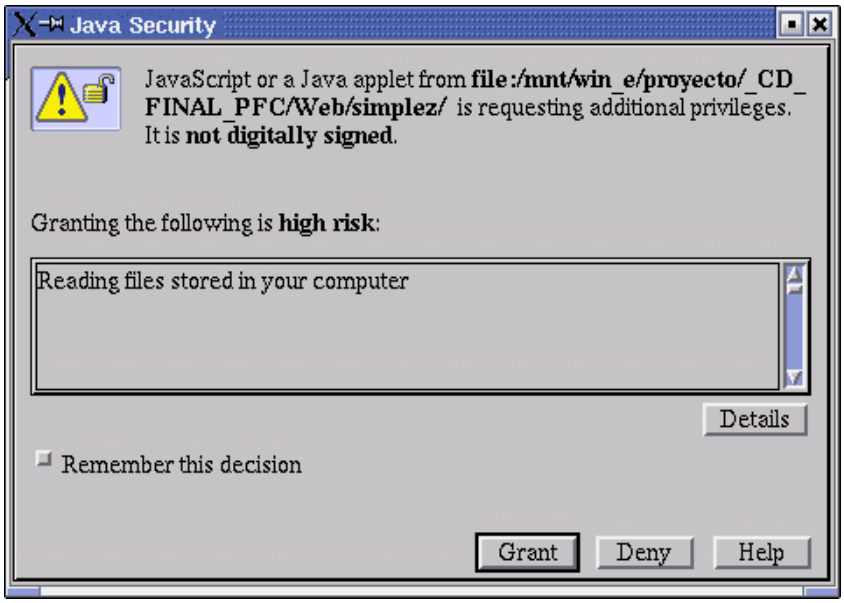
Esta ventana solo se proporciona como un método cómodo de acceder al menú de carga de programas, independiente de la ventana del navegador usado (para el caso de la versión en applet).



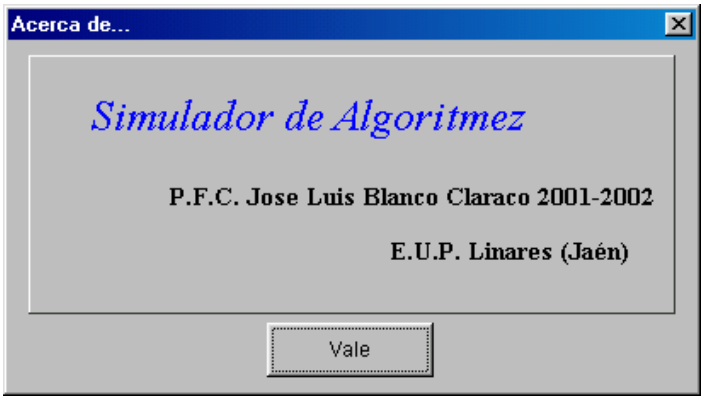
Contiene los siguientes menús:

- Ensamblador
  - Nuevo programa: Abre una ventana de editor de programas en blanco.

- Cargar programa: Permite cargar un programa guardado en disco. Si se está usando Netscape, aparecerán dos ventanas solicitando derechos de lectura y escritura en disco, que deberán aceptarse (“Grant”) si se quiere cargar o grabar programas en ficheros:

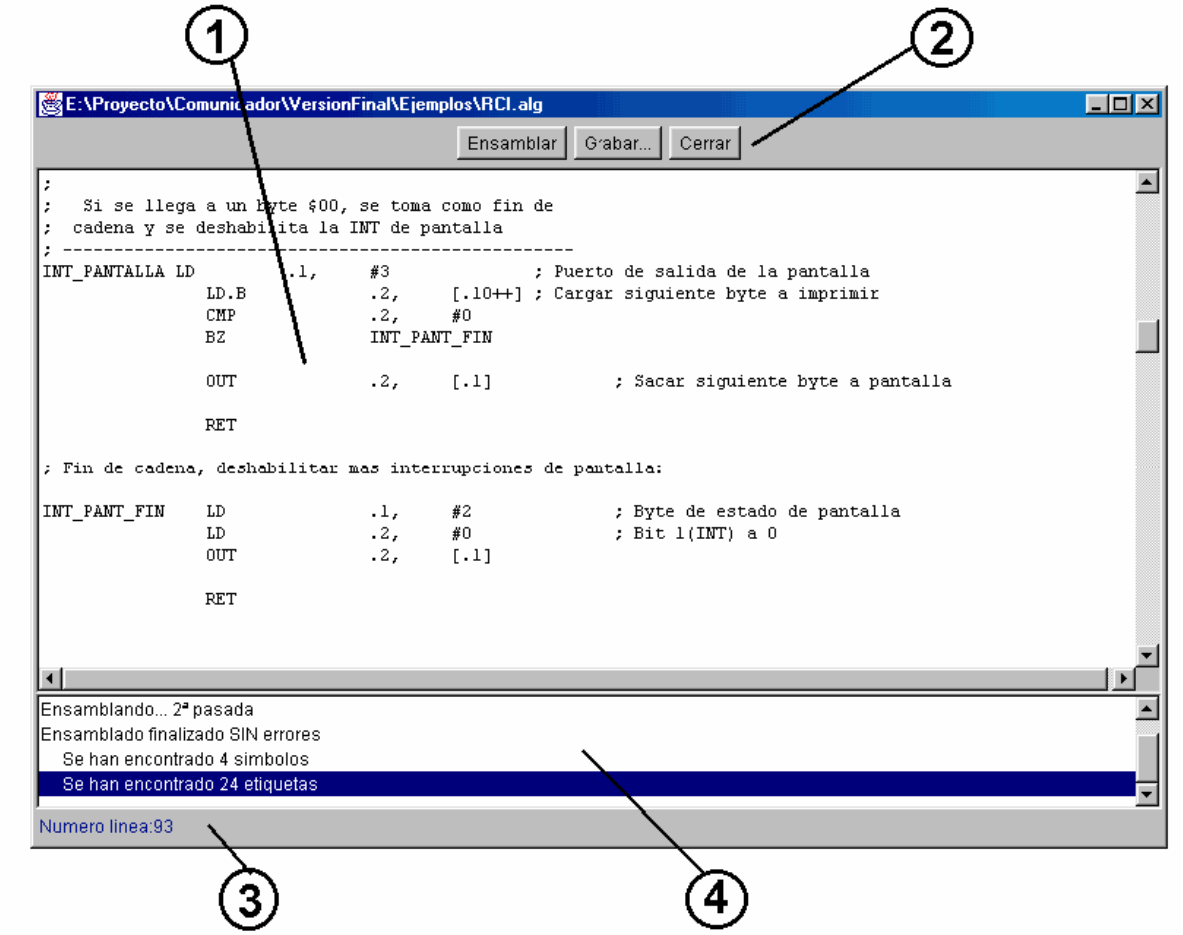


- Ayuda
  - Ayuda de Algorítmez: Solo disponible si se ha usado la versión online, abre una ventana del explorador con esta guía de usuario.
  - Acerca de: Muestra información sobre el programa.



A5.2.3. Editor de programas

El editor de programas sirve para la edición de los programas Algorítmez, ensamblarlos y lanzar la simulación de los mismos:



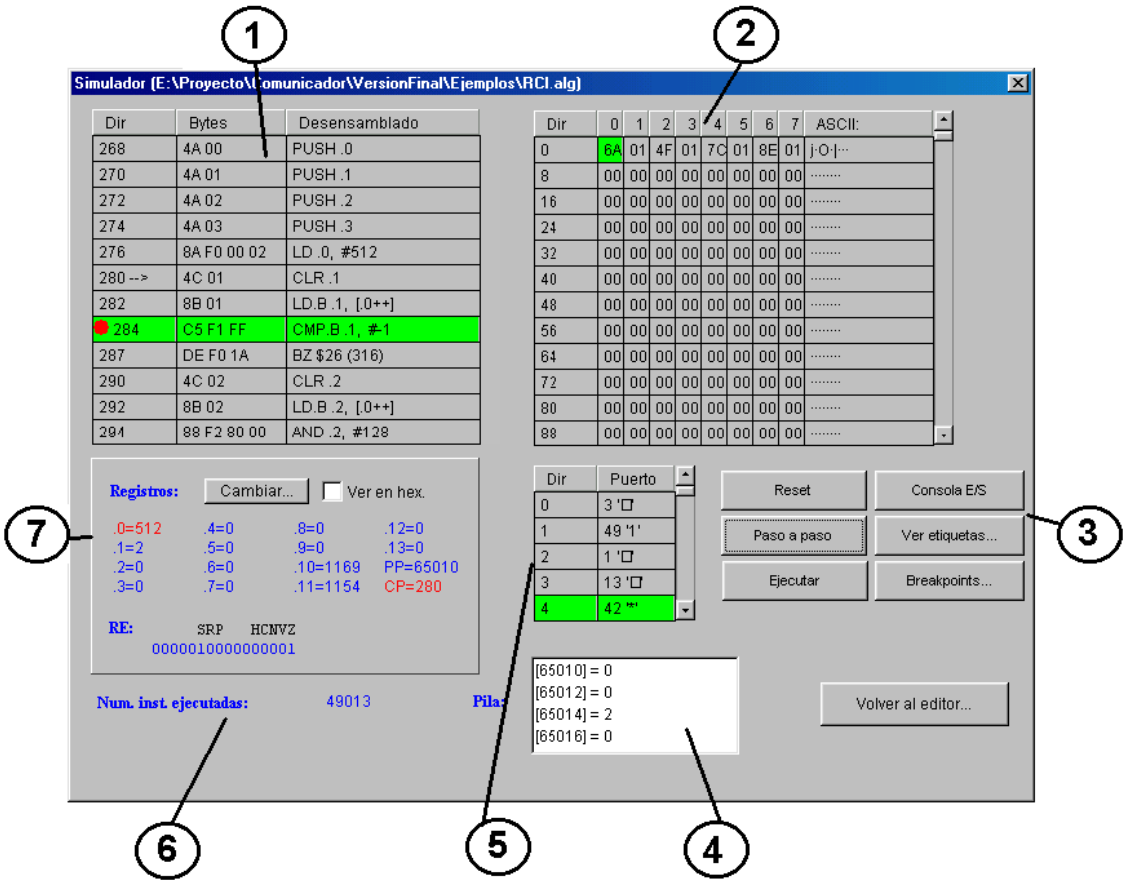
Se pueden ver las siguientes partes:

1. Campo del editor, donde se puede ver y editar el programa.
2. Barra superior de botones. Los botones son los siguientes:
    - Ensamblar y simular: Ensambla el programa y muestra los mensajes del ensamblador en la zona de mensajes inferior. En caso de no producirse ningún error, se abrirá la ventana del simulador con el programa cargado en memoria y listo para iniciar la simulación.
    - Grabar. Permite guardar el programa en un fichero. Es preciso haber aceptado los permisos solicitados si se ejecuta desde un navegador para que esta opción este disponible.
    - Salir: Cierra esta ventana del editor.

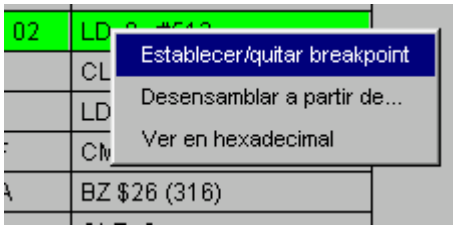
3. Barra inferior. Aquí se muestra el número de línea sobre la que se encuentra el cursor en cada momento.
4. Zona de mensajes, donde se muestran los mensajes y los errores generados por el ensamblador. Haciendo doble-clic sobe un error, se resaltará automáticamente la línea correspondiente al error en el programa. Esta lista se puede esconder si se desea pulsando el botón derecho sobre el campo de texto del editor, y seleccionando en el menú contextual la opción "Mostrar/ocultar mensajes del ensamblador".

A5.2.4. Ventana del simulador

Tras ensamblar correctamente un programa Algorítmez, se abrirá la ventana principal del simulador, con el programa cargado y listo para su simulación:



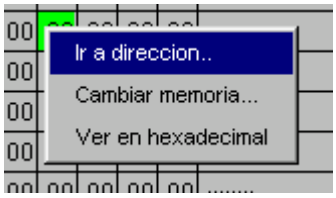
1. Zona de desensamblado: Aquí se ven las próximas instrucciones a ejecutar desensambladas. Pulsando sobre la tabla con el botón derecho del ratón, aparece este menú contextual:



Donde:

- "Establecer/quitar breakpoint", marca o quita la dirección marcada como un breakpoint o punto de ruptura. Estos puntos paran la ejecución del simulador cuanto el CP llegue a la dirección señalada, justo antes de ejecutar esa instrucción.
- "Desensamblar a partir de..." permite introducir una dirección de memoria que se quiera ver como programa desensamblado. En esta petición de direcciones, así como en todas las demás del simulador, se pueden introducir tanto valores decimales, hexadecimales, octales, o incluso etiquetas del programa.
- "Ver en hexadecimal", casilla que se puede marcar o desmarcar, con lo que las direcciones y los operandos se mostrarán en hexadecimal o decimal, respectivamente.

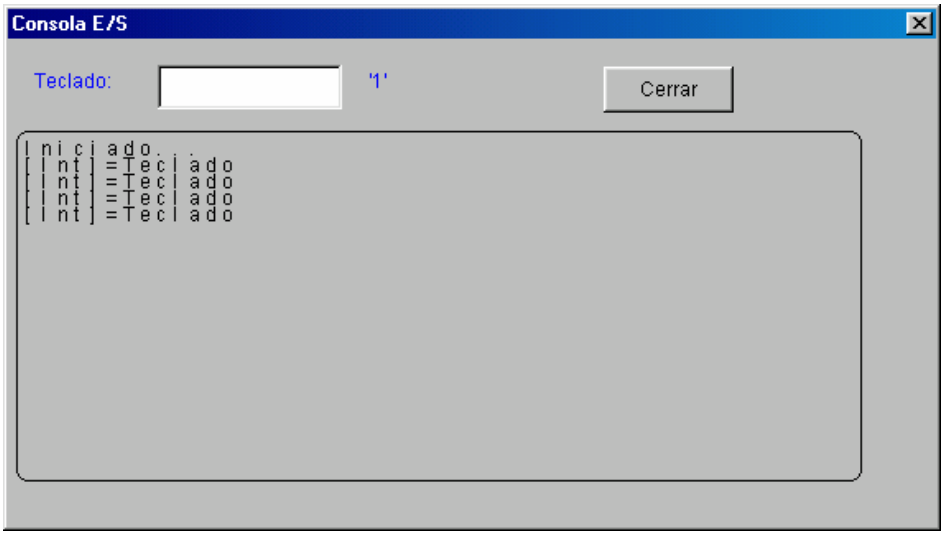
2. Vista de memoria. Aquí se puede ver el contenido de las 64 Kbytes de memoria de Algorítmez, en hexadecimal y como códigos ASCII. Pulsando con el botón derecho aparece este menú contextual:



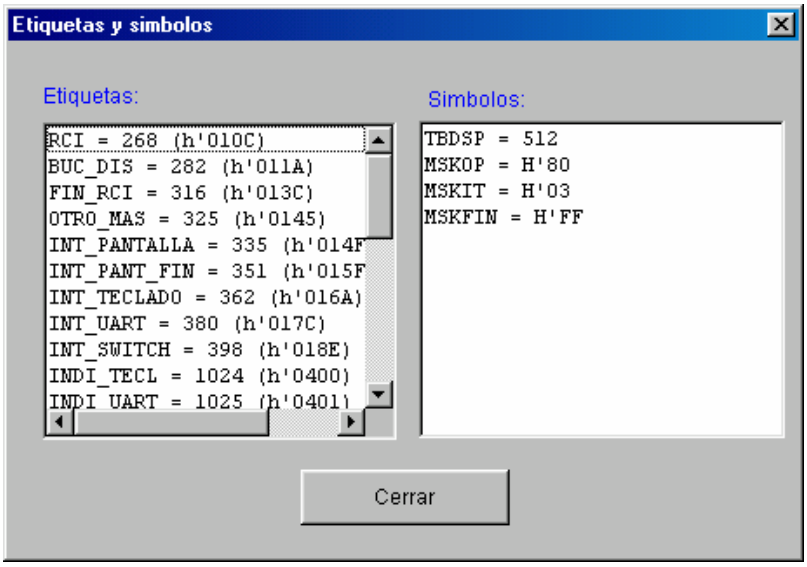
Las operaciones que se pueden realizar desde aquí son:

- “Ir a dirección...”, realiza el *scroll* necesario en la tabla para que la dirección dada sea visible.
- “Cambiar memoria”, permite introducir un nuevo valor para una dirección de RAM.
- “Ver en hexadecimal”. Marcando o desmarcando esta casilla se muestran las direcciones de la tabla en hexadecimal o en decimal, respectivamente.

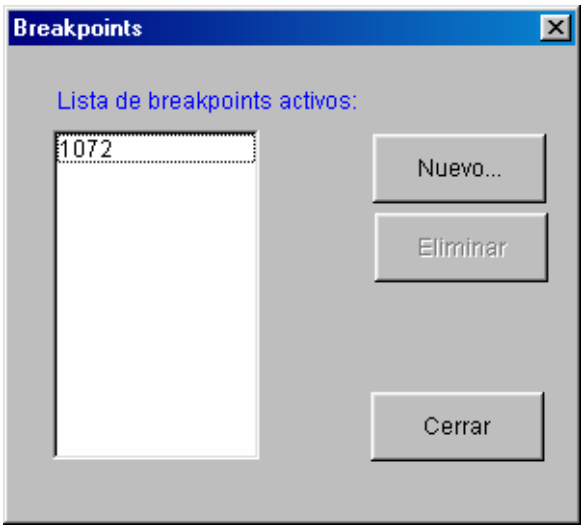
3. Botones de control y útiles: Son los siguientes:
    - “Reset”: Reinicia el simulador de Algorítmez, recargando la imagen de RAM original que resultó tras ensamblar.
    - “Paso a paso”: Ejecuta una sola instrucción.
    - “Ejecutar”: Inicia el modo de ejecución continuo. Al pulsarlo, este botón se convierte en el botón “PARAR”, que finaliza manualmente el modo de ejecución continuo. También se puede para este modo por llegar a un HALT o a un punto de ruptura.
    - “Consola E/S”: Muestra la consola de entrada y salida, que implementa los periféricos de Algorítmez pantalla y teclado:



- “Ver etiquetas”, muestra una lista con todas las etiquetas y símbolos declarados en el código:



- “Breakpoints”, permite editar la lista de puntos de ruptura del simulador. Esta lista siempre se pierde al cerrar la ventana del simulador.



4. Pila: Muestra las últimas palabras (2 bytes) introducidas en la pila.
5. Puertos: En esta lista se ve el contenido de los puertos de Algorítmez.
6. Contador de instrucciones: Aquí se muestra el valor del contador de instrucciones ejecutadas desde el reset.
7. Registros. Se muestra el estado de todos los registros, señalando en rojo los que han cambiado en el último paso, o en el caso del registro de estado, los bits que

han cambiado. Pulsando “Cambiar” se abre una ventana desde donde se pueden modificar los valores de los registros:

A5.2.5. Referencia rápida de Algorítmez

Lenguaje ensamblador

- Directivas: Son sentencias que sólo sirven para dar órdenes al programa ensamblador y que no generan contenido alguno en la memoria. El ensamblador de Algorítmez soporta las siguientes:

  - ORG <Constante 16 bits>

    Indica al programa ensamblador a partir de que dirección de la memoria tiene que guardar las instrucciones o pseudoinstrucciones que le siguen.

  - END <Constante 16 bits>

    Indica al programa ensamblador que finaliza el código fuente e indica la dirección del punto de entrada para el programa.

- <Nombre> EQU <Valor>

  Declara un nuevo símbolo y su valor correspondiente. Siempre que el ensamblador encuentre el nombre del símbolo en una línea de código, lo sustituirá por su valor.

- Pseudoinstrucciones: Son sentencias que aún no siendo instrucciones del lenguaje ensamblador, sí afectan al contenido de la memoria durante el ensamblado:
  - RES <Constante> y RES.B <Constante>

    Reserva un número de bytes igual al indicado, si se trata de RES.B o el doble, si de trata de RES.

  - DATA.B <Constante> [, <Constante> [, <Constante>, ...]]

    Genera directamente en memoria los valores indicados de un byte. Cada constante debe ir separada con una coma. Se pueden usar cadenas de texto encerradas en dobles comillas (“) con lo que se generará la secuencia de bytes en código ASCII correspondiente.

  - DATA <Constante> [, <Constante> [, <Constante>, ...]]

    Igual que DATA.B con la diferencia de que en éste se permiten constantes de dos bytes. Aunque alguna constante se pudiese representar con solo un byte, o sean caracteres, se representarán de igual manera ocupando dos bytes.

- Instrucciones: Son los mnemónicos que representan las correspondientes operaciones máquina del microprocesador. Cada línea de programa puede contener una sola instrucción. El formato general de una línea de programa con una instrucción es:

TIPO 0: [Etiqueta] <mnemónico>
TIPO 1: [Etiqueta] <mnemónico> .R [,.RX]
TIPO 2: [Etiqueta] <mnemónico> .R, <mem>
TIPO 3: [Etiqueta] <mnemónico> <mem>

Donde:

- Corchetes ([ ]) indican que no siempre es obligatorio.
- .R o .RX son registros. P.ej: .0, .15 , etc...
- <mem> representa alguno de los 8 modos de direccionamiento a memoria de que dispone Algorítmez. A continuación se presenta la sintaxis que debe tener cada uno:

Modo	Sintaxis	Comentarios
Autoincremento	`[.RX++]`	La dirección efectiva es el contenido del registro, antes de incrementarlo.
Indexado	`/<8BITS>[.RX]`	Desplazamiento de 8 bits con signo respecto a un registro.
Autoincremento Indirecto	`[[.RX++]]`	La dirección eficaz es el valor de 16 bits almacenado en la dirección de MP indicada por RX.
Indexado indirecto	`[/<8BITS>[.RX]]`	Desplazamiento de 8 bits con signo respecto a un registro e indirección.
Inmediato	`#<8 o 16 BITS>`	Valor inmediato. Longitud depende de la instrucción.
Relativo a programa	`<16BITS>` ó `$<16BITS>`	La dirección de 16 bits indicada debe estar en el rango: CP-127 a CP+128
Directo	`/<16BITS>`	Dirección de 16 bits absoluta.
Relativo a programa e indirecto	`[<16BITS>]` ó `[$<16BITS>]`	La dirección de 16 bits indicada debe estar en el rango: CP-127 a CP+128

- La etiqueta (optativa) asociará un nombre a la dirección en memoria, facilitando su referenciación. Las etiquetas también se pueden usar con las seudo-instrucciones, siendo la dirección asociada a la etiqueta en cualquier caso la dirección del contador de direcciones del ensamblador antes de procesar la línea.

- Comentarios: Independientemente del contenido de una línea, se pueden añadir comentarios usando el carácter punto y coma (“;”), con lo que el ensamblador ignorará el contenido del resto de la línea.

Instrucciones

Estas son las instrucciones de Algorítmez. Se marcan con un símbolo (*) las que son privilegiadas, es decir, sólo pueden ser ejecutadas (sin dar lugar a una interrupción de violación de modo) en modo supervisor:

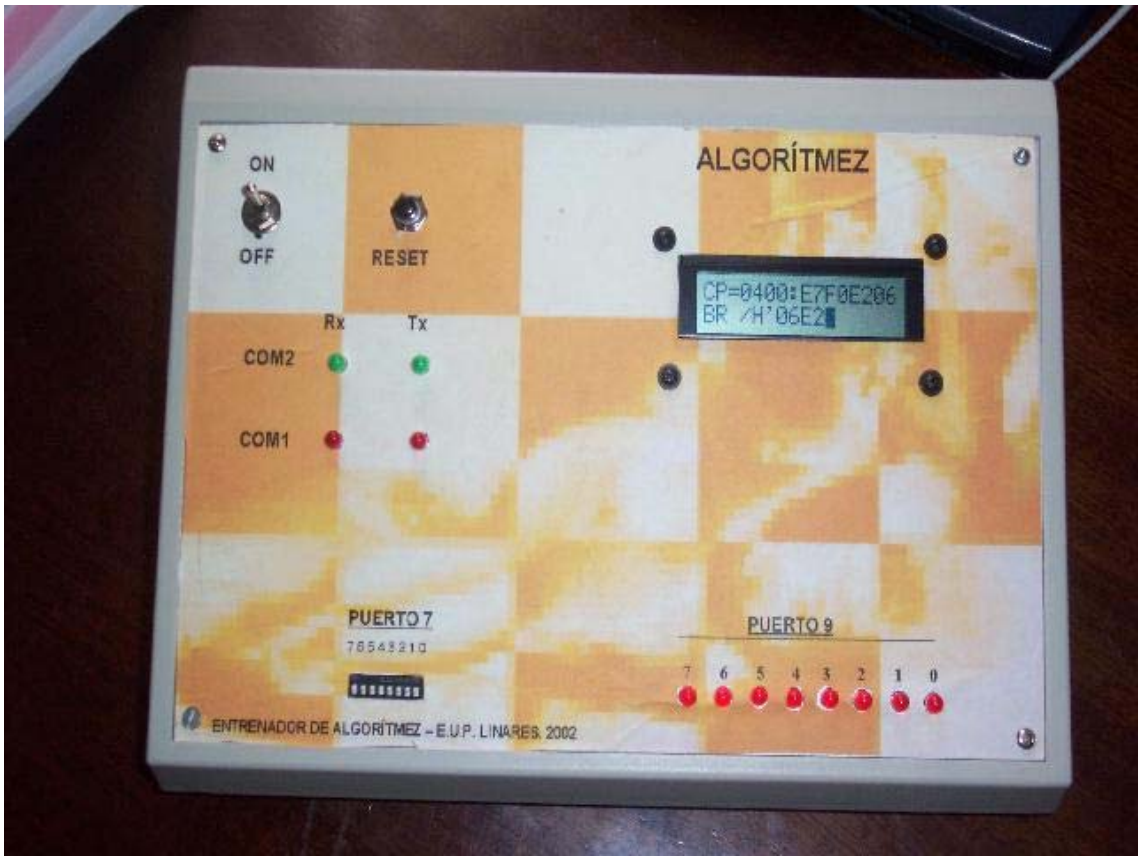
OPCODE / TIPO	0	1	2	3
$0	CLRC	SHR	ADD	LD.E (*)
$1	CLRV	SHL	ADD.B	ST.E
$2	EI (*)	ROR	ADDC	LD.B.E
$3	DI (*)	ROL	ADDC.B	ST.B.E
$4	BRK	RORC	SUB	CMP
$5	BRKV	ROLC	SUB.B	CMP.B
$6	NOP	SHRA	SUBC	CALL
$7	WAIT (*)	SHLA	SUBC.B	BR
$8	HALT (*)	IN (*)	AND	BC
$9	RET	OUT (*)	OR	BNC
$A	RETI (*)	PUSH	LD	BV
$B	*X*	POP	LD.B	BNV
$C	*X*	CLR	ST	BN
$D	*X*	NOT	ST.B	BNN
$E	*X*	NEG	*X*	BZ
$F	*X*	ADJ	*X*	BNZ

Periféricos

Los dos periféricos, pantalla y teclado, tienen asignados cada uno dos puertos de E/S, con el siguiente uso:

Puerto	Periférico	Uso
0	Teclado	P. de estado. Bit 0: Preparado. Bit 1: Permitir interrupciones
1		Entrada: Carácter de tecla pulsada.
2	Pantalla	P. de estado. Bit 0: Preparado. Bit 1: Permitir interrupciones
3		Salida: Carácter a imprimir.

A5.3. Entrenador de Algorítmez

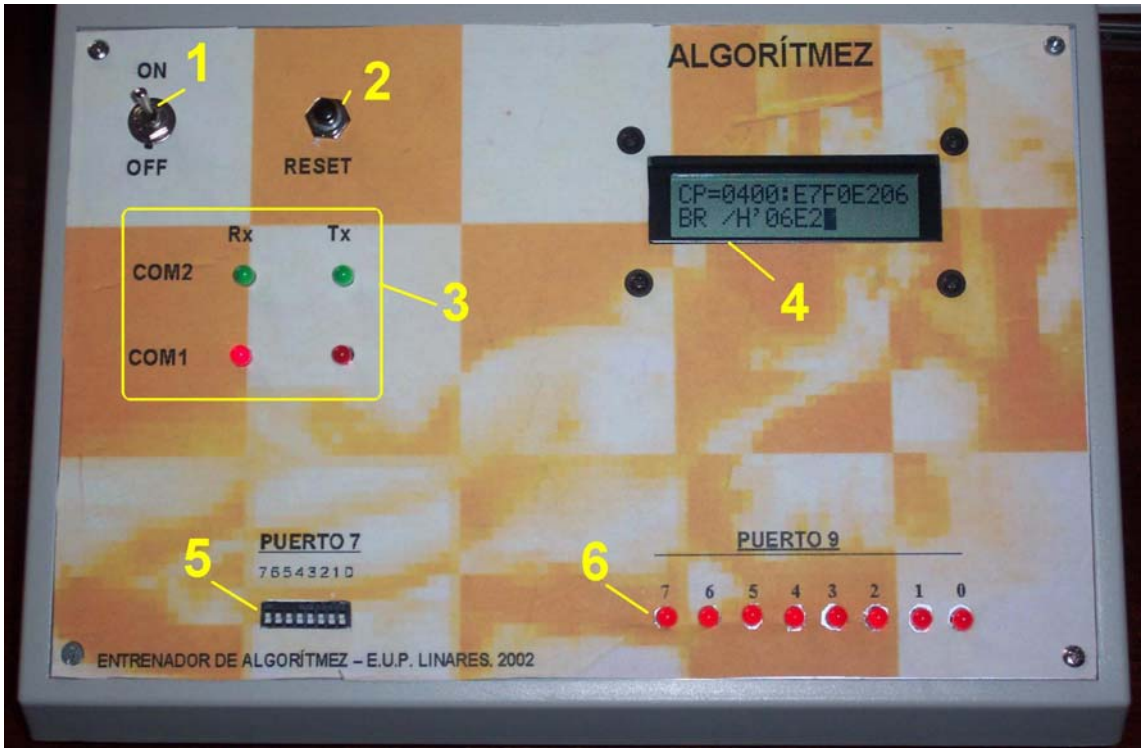


A5.3.1. Introducción

El entrenador es un sistema hardware que de forma independiente a un PC es capaz de ejecutar programas para Algorítmez. Como este procesador no existe comercialmente, se ha usado un microcontrolador de propósito general, el AT90S8515, y se ha programado de tal forma que se comporte como si fuera un microprocesador Algorítmez. Este microcontrolador tiene dos modos de funcionamiento principales: El modo depuración y el modo de solo ejecución. En este último realmente se comporta sólo como un microprocesador Algorítmez, mientras que en el primero, usa la pantalla LCD y el teclado principalmente para mostrar al usuario un sistema de menús desde el que se puede ejecutar paso a paso, ver y modificar el contenido de los registros, entre otras posibilidades.

A5.3.2. Periféricos y conexiones externas

En el panel frontal del entrenador es donde están la mayoría de periféricos y controles de éste:



1. Interruptor de encendido y apagado.
2. Pulsador de reset de sistema.
3. Indicadores LEDs de estado de puertos serie. Cuando se iluminan indican actividad en el puerto COM1 o COM2. Se muestran separadamente las líneas Rx (recepción) y Tx (transmisión).
4. Pantalla LCD. Esta es usada tanto para mostrar los menús del modo depuración como para mostrar el contenido de la pantalla virtual de Algorítmez.
5. Interruptores. Constan de 8 microswitchs, que están cableados al nuevo puerto de entrada de Algorítmez.
6. Conjunto de 8 indicadores LEDs. Cableados al nuevo puerto de salida de Algorítmez.

En la parte posterior del entrenador se puede observar de izquierda a derecha: Conectores DB9 para puerto serie RS232 correspondientes a los puertos COM1 y COM2, conector PS/2 para teclado estándar de PC y cable de alimentación.

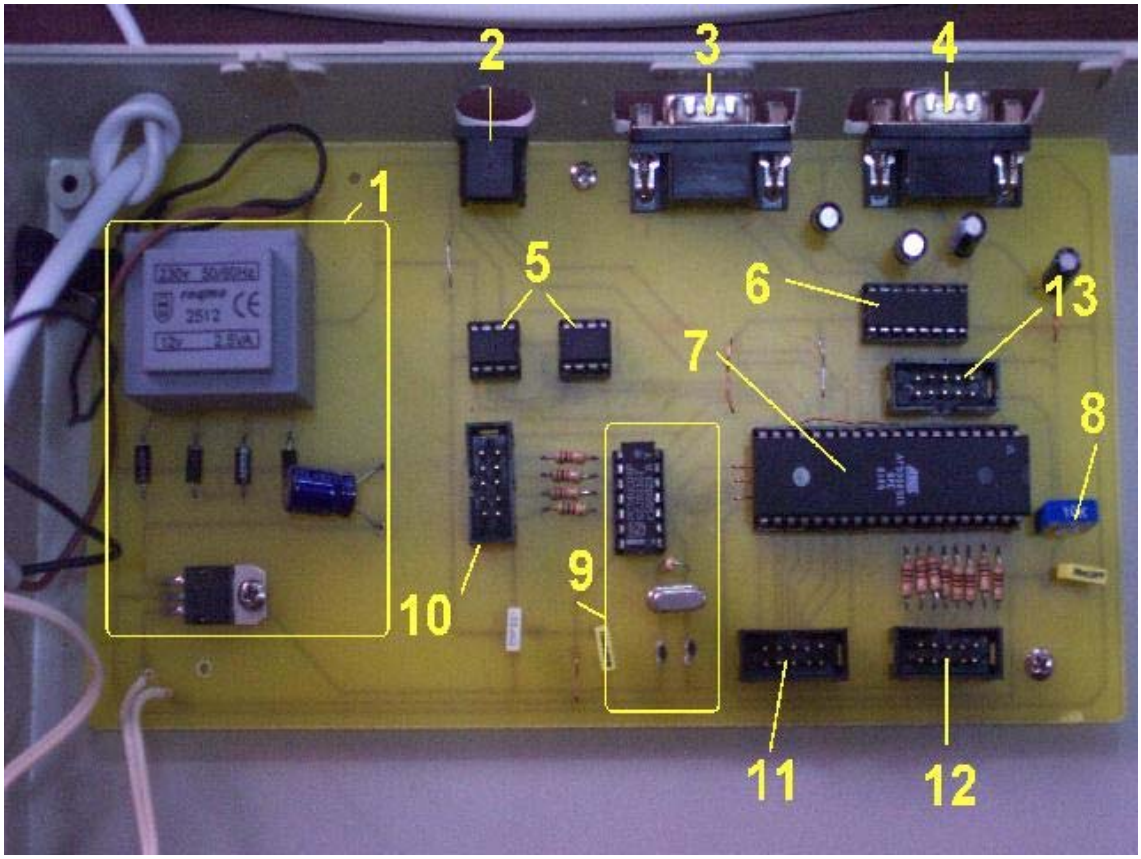
El puerto serie COM1 es usado exclusivamente para comunicaciones con el programa Comunicador, para realizar tareas como la descarga de programas Algorítmez o el depurado remoto. Se usa siempre una configuración de 115200 bps, 8 bits de datos, sin paridad y sin ningún protocolo de control de flujo.

En cuanto al puerto COM2, es un puerto serie accesible por los programas Algorítmez que se ejecuten en el entrenador. La velocidad se puede configurar desde programas Algorítmez, aunque siempre se usarán 8 bits de datos y sin paridad.

El puerto de teclado PS/2 es compatible con cualquier teclado convencional para PCs. El teclado se usará exclusivamente como teclado de Algorítmez si el entrenador está en modo ejecución. Si se está en modo depuración, se usará para moverse por los distintos menús, y sólo actuará como teclado de Algorítmez cuando se entre en modo ejecución continua.

A5.3.3. Conexiones internas

El entrenador está formado por el panel frontal que se ha visto anteriormente, la caja de alojamiento y la placa del sistema, que se conecta con el panel frontal por medio de cables planos:



A continuación se listan las partes que se pueden observar en el sistema. Los esquemas completos y fotolitos se pueden descargar de la web del proyecto.

1. Fuente de alimentación.
2. Conector para teclado PS/2.

3 y 4. Conectores para puertos serie COM 1 y COM 2.

5. Bancos de memoria 0 y 1. Cada uno es un chip AT24C512, con 32 Kbytes de memoria EEPROM. Los dos están conectados al mismo bus I2C.
6. Chip MAX232, para la conversión de los niveles lógicos TTL a RS232.
7. Microcontrolador central, un AT90S8515.
8. Potenciómetro para ajuste del contraste del display.

9. Oscilador a cristal, a frecuencia de 7.3728 Mhz.
10. Conector de cable plano. A los LEDs indicadores de los puertos series y pulsador de reset.
11. Conector de cable plano. Entrada al puerto 5 de Algorítmez. Conectado a los microswitchs en el panel frontal.
12. Conector de cable plano. Salida del puerto 7 de Algorítmez. Conectado a los diodos LEDs del panel frontal.
13. Conector de cable plano. Para el display LCD del panel frontal.

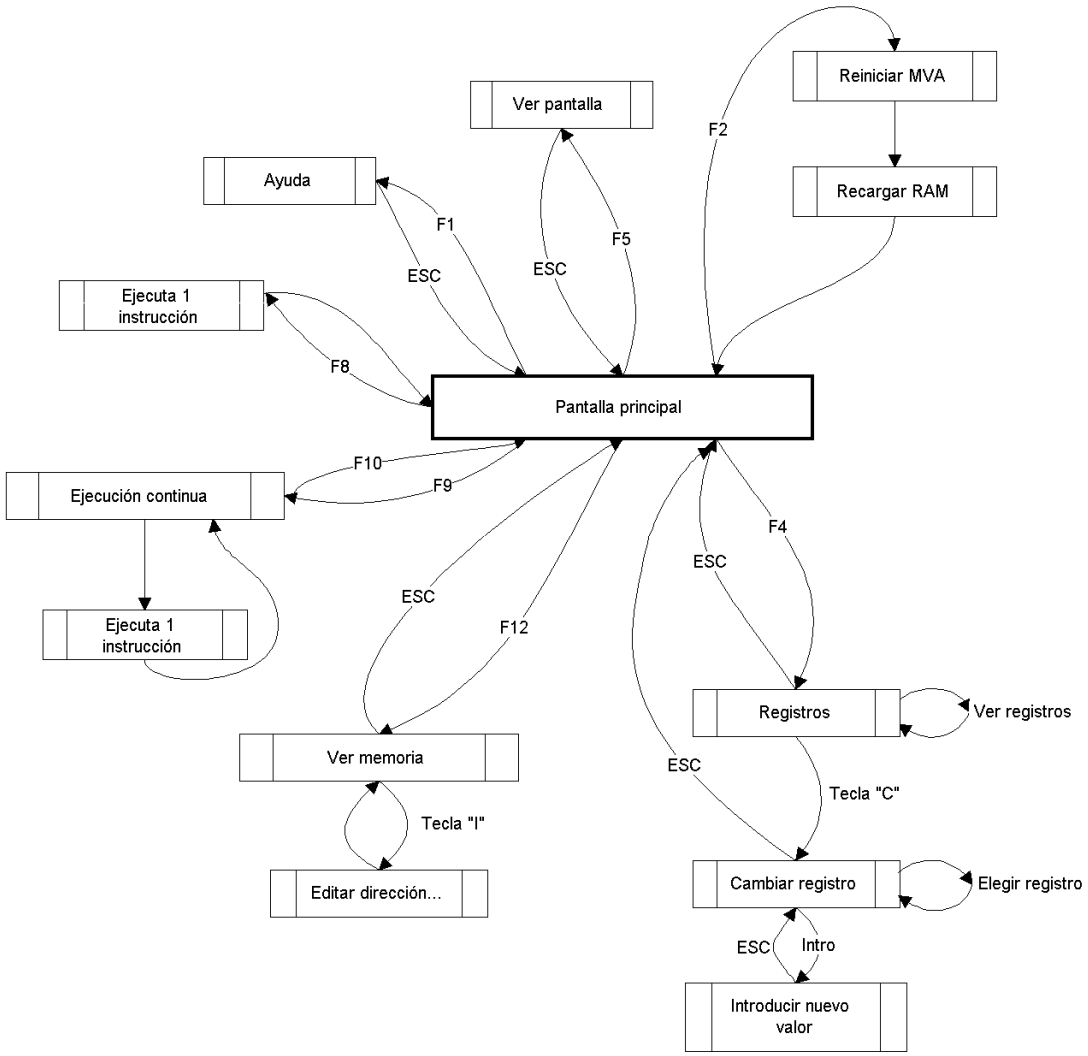
A5.3.4. Funcionamiento de la memoria

Existen dos bancos de memoria, cada uno de 64Kb. En el banco 0 (el que tiene 0 por dirección de bus I2C), se usa para la almacenar la memoria RAM de Algorítmez en cada momento durante la ejecución de los programas. El segundo banco, el banco 1, es donde se guarda la imagen original de RAM con el programa Algorítmez a emular, que se envía desde el programa Comunicador del PC. Cada vez que se reinicia la máquina virtual Algorítmez del entrenador, se recargan las 64Kbs desde el banco 1 al banco 0, formando la imagen original del programa.

Además, esto implica que el programa se queda almacenado aún en ausencia de alimentación, al usar una memoria EEPROM.

A5.3.5. Sistema de menús

Este es el esquema de menús implementado en el entrenador de Algorítmez:



Se parte de la pantalla principal (siempre en modo depuración, ya que en modo ejecución no se puede acceder a ningún menú) y se muestra sobre los enlaces la tecla con la que se accede a cada menú. En la mayoría de los casos la tecla escape (ESC) sirve para salir del menú actual y volver al anterior.

- Pantalla principal: Aquí se muestra desensamblada la siguiente instrucción a ejecutar, como se ve en la primera foto del manual.

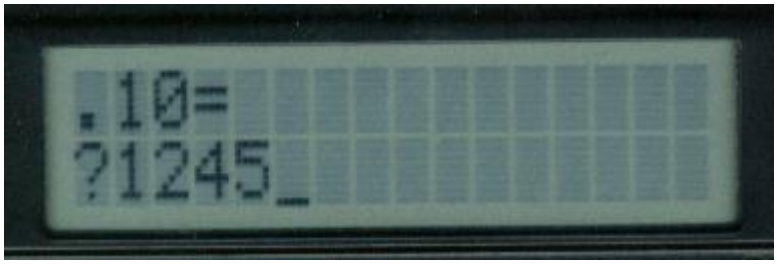
- Ayuda: Pulsando F1 se muestra la pantalla de ayuda, donde pulsando las teclas de flecha arriba y flecha abajo se hace scroll por las líneas del texto, que enumera las teclas con las que se entra a cada menú:

- Reset de MVA (Máquina virtual Algorítmez): Pulsando F2 se reinicia el estado de Algorítmez y se recarga la memoria del banco 0 con el contenido del banco 1, es decir, con la copia original del programa a ejecutar:

- Ver/editar registros: Pulsando F4 se muestra el visor de registros, donde se puede ver el listado de los 16 registros y su contenido (en hexadecimal), además del contenido del registro de estado, este último en binario y con una leyenda del significado de cada bit. Para moverse por la lista hay que usar las teclas de flecha arriba y flecha abajo:

Pulsando la tecla “C” en este menú se entra en el menú de edición de registros, donde se permite cambiar el contenido de cualquiera de ellos. Aparece una lista de los registros donde con las flechas de arriba y abajo se selecciona el registro que se quiere cambiar:

Y pulsando Intro sobre este, se nos pide que introduzcamos por teclado el nuevo valor en hexadecimal:



- Ver pantalla de Algorítmez: Pulsando F5 se muestra el contenido actual de la pantalla de Algorítmez, hasta que se pulse ESC para volver a la pantalla principal:

- Ejecutar un paso: Pulsando F8 se ejecuta sólo una instrucción y se muestra la nueva siguiente instrucción desensamblada.

- Iniciar ejecución continua: Pulsando F10 se inicia el modo de ejecución continua. En este modo se muestra el estado de la pantalla virtual de Algorítmez y se ejecutan instrucciones continuamente, hasta que se pulse F9 para parar, o se llegue a un punto de ruptura, o se reciba del comunicador un comando de parar ejecución. Al parar se vuelve a la pantalla principal mostrando la nueva siguiente instrucción a ejecutar.

- Visor de memoria: Pulsando F12 se entra a la vista de memoria:

A pesar de las limitaciones de tamaño del LCD se ha conseguido mostrar los siguientes campos por línea: Dirección, contenido (3 bytes por línea) y contenido ASCII (esos 3 bytes mostrados como caracteres).

En este modo, se muestra el cursor y se puede mover usando las cuatro flechas de dirección, además de las dos teclas de avanzar página y retroceder página. Si se sobrepasan tanto el final como el comienzo de una línea, se realizan las operaciones de *scroll* necesarias. Además, pulsando la tecla "I" se puede ir directamente a una dirección determinada, introduciéndola por teclado.

A5.3.6. Diferencias con simulador Algorítmez

Mientras el simulador software de Algorítmez creado también en este proyecto, se ha diseñado lo más exactamente posible a la especificaciones dadas por Gregorio Fernández, en el entrenador se han realizado unas pequeñas modificaciones, debido a las nuevas posibilidades que esta plataforma proporciona.

Memoria:

Las 64Kbytes de memoria son de tipo RAM y puede suponer que tras el reset ya contiene el programa ensamblado completo, a diferencia de la especificaciones de Gregorio Fernández donde existía una zona alta de ROM.

Puertos:

Se han añadido nuevos puertos accesibles desde programas Algorítmez:

Puerto	Periférico	Uso
0	Teclado	P. de estado. Bit 0: Preparado. Bit 1: Permitir interrupciones
1		Entrada: Carácter de tecla pulsada.
2	Pantalla	P. de estado. Bit 0: Preparado. Bit 1: Permitir interrupciones
3		Salida: Carácter a imprimir.
4	UART	P. de estado. Bit 0: Byte recibido. Bit 1: Permitir interrupciones (Ver abajo más explicaciones)
5		Entrada y salida: Byte recibido o byte a enviar, respectivamente.
6	Switchs	P. de estado. Bit 0: Preparado. Bit 1: Permitir interrupciones
7		Entrada: Estado de los 8 switchs.
8	LEDs	No usado
9		Salida: Nuevo estado de los 8 LEDs.

Cualquier salida a un puerto no definido será ignorada, así como las lecturas de puertos no definidos o de sólo escritura, se leerán como cero.

Se ha guardado compatibilidad con los dos periféricos de los emuladores software, el teclado y la pantalla. La UART emulada por software permite la comunicación de programas en Algorítmez con cualquier otro equipo a través del puerto COM2 del entrenador. Los LEDs y los switchs son dispositivos instalados en el entrenador de forma permanente y pretenden servir de puertos de salida y entrada, respectivamente, fácilmente visibles y manipulables manualmente por el usuario para la depuración de aplicaciones. A continuación se ve en más detalle el uso de cada puerto, el significado de cada bit y si el puerto es de entrada y salida (E/S), solo entrada (E) o solo salida (S).

Puerto[0]: Teclado – Estado (E/S)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	X						INT	PR
Valor inicial	0						0	0

Solo se puede escribir el bit INT, ignorando la escritura en los demás bits.

- PR: Bit de preparado. Cuando esta a 1, significa que se ha pulsado una tecla y esta lista para ser leída.
- INT: Máscara de interrupciones. Si esta a 0 inhibe la petición de interrupción de éste dispositivo cuando su estado sea de preparado.

Puerto[1]: Teclado – Datos (E)

Bit	7	6	5	4	3	2	1	0
Significado	CARÁCTER							
Valor inicial	0x00							

Al leer este puerto se obtiene el código ASCII del carácter que se ha pulsado en el teclado. Ver las limitaciones respecto a los caracteres leídos desde el teclado en el punto 4.3.5.4.
Al realizar una operación de lectura de éste puerto, se pone a cero automáticamente el bit PR del puerto de estado del teclado.

Puerto[2]: Pantalla – Estado (E/S)

Bit	7	6	5	4	3	2	1	0
Significado	X						INT	PR
Valor inicial	0						0	1

Solo se puede escribir el bit INT, ignorando la escritura en los demás bits.

- PR: Bit de preparado. Cuando esta a 1, significa que el periférico esta preparado para enviar un nuevo carácter a la pantalla.
- INT: Máscara de interrupciones. Si esta a 0 inhibe la petición de interrupción de éste dispositivo cuando su estado sea de preparado.

Puerto[3]: Pantalla – Datos (S)

Bit	7	6	5	4	3	2	1	0
Significado	CARÁCTER							
Valor inicial	0x00							

Al escribir en este puerto, se envía el byte como carácter a imprimir a la pantalla virtual de Algorítmez. Tras hacer esto se pone a cero el bit de preparado del puerto de estado de la pantalla, que se volverá a poner a uno tras un período de tiempo, que se ha elegido de 50 instrucciones ejecutadas.

Si se envía un carácter 0x00 a la pantalla, esta se limpiará y se colocará el cursor en la posición inicial. Si se envía el carácter 0x01 se realizará un scroll de una línea hacia arriba. Si se envía un carácter normal y el cursor esta al final de una línea, se saltará automáticamente al comienzo de la siguiente, así como se realizará scroll de pantalla si es necesario.

Puerto[4]: UART – Estado (E/S)

Bit	7	6	5	4	3	2	1	0
Significado	X	VEL					INT	PR
Valor inicial	0	0x00					0	1

Se puede escribir en los bits 1 a 6, ignorando la escritura en los demás bits.

- PR: Bit de preparado. Cuando esta a 1, significa que se ha recibido un carácter por el puerto serie COM2, y este está almacenado en el buffer accesible leyendo el puerto de datos.
- INT: Máscara de interrupciones. Si esta a 0 inhibe la petición de interrupción de éste dispositivo cuando su estado sea de preparado.
- VEL: Velocidad de bit. La relación entre la velocidad de bit que usa la UART para recibir y enviar, y el campo VEL se ha programado de la siguiente forma:

$$VEL = \frac{7376800}{BAUD \cdot 64} - 2 \qquad BAUD = \frac{7376800}{(VEL + 2) \cdot 64}$$

Donde 7376800 = 7.3768 MHz es la frecuencia del cristal con el que funciona el microcontrolador, escogida especialmente para ser múltiplo de todas las frecuencias estándar de transmisión serie y BAUD es la velocidad de bit en bits por segundo.
La siguiente tabla contiene ya calculados los valores de VEL para valores estándar de BAUD:

Velocidad (bps)	**Valor de "VEL"**
57600	0
38600	1
19200	4
9600	10
4800	12

Puerto[5]: UART – Datos (E/S)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Significado	BYTE							
Valor inicial	0x00							

Si se lee este puerto, se obtiene el último byte recibido por el puerto serie COM2 del entrenador. Sólo se debería leer este puerto si se ha comprobado antes que ha llegado un byte nuevo, lo que se verá en el bit preparado del puerto de estado que deberá estar a uno. Al realizar la lectura sobre este puerto, se borra automáticamente el bit de preparado del puerto de estado.

Si se escribe en este puerto, se envía inmediatamente el byte por el puerto serie COM2 a la velocidad programada en el puerto de estado. Debido a las limitaciones de la UART emulada por software, se para la ejecución del programa Algorítmez mientras se está enviando este byte, lo que no debería ser un problema ya que por ejemplo a 57600bps eso representa una parada de solo 134μs.

Puerto[6]: Switchs – Estado (E/S)

Bit	7	6	5	4	3	2	1	0
Significado	X						INT	PR
Valor inicial	0						0	0

Solo se puede escribir el bit INT, ignorando la escritura en los demás bits.

- PR: Bit de preparado. Cuando esta a 1, significa que se ha detectado un cambio en el estado de los switchs.
- INT: Máscara de interrupciones. Si esta a 0 inhibe la petición de interrupción de éste dispositivo cuando su estado sea de preparado.

Si el estado tras el reset de los switchs es distinto a 0x00, el bit de preparado se activará inmediatamente. Para que se dispare el bit de preparado de este puerto, el cambio en el estado de los switchs debe durar más que el tiempo de ejecución de una instrucción Algorítmez.

Puerto[7]: Switchs – Datos (E)

Bit	7	6	5	4	3	2	1	0
Significado	Estado de los 8 micro-switchs							
Valor inicial								

Una lectura en este puerto conlleva una lectura física inmediata del puerto C del microcontrolador, cableado en el entrenador a un micro switchs de 8 interruptores. Estos deben hacer contacto con masa al activarse, ya que se han activado internamente las resistencias de tirón a Vcc en cada uno de los pins del puerto, de forma que al no cerrarse un interruptor, ese bit se lee como un uno.
Al leer este puerto se pone a cero el bit de preparado del puerto de estado.

Puerto[8]: LEDs – Estado (E)

Bit	7	6	5	4	3	2	1	0
Significado	X							
Valor inicial	0							

Este puerto no se usa y siempre es leído como 0x00.

Puerto[9]: LEDs – Datos (S)

Bit	7	6	5	4	3	2	1	0
Significado	Valor a mostrar en los 8 leds							
Valor inicial	0x00							

Al escribir un byte en este puerto se saca este inmediatamente por el puerto A del microcontrolador, donde en el entrenador se han colocado 8 diodos LEDs, cableados de forma que se enciendan cada uno al poner un 1 en su bit correspondiente.

Tabla de periféricos:

Al añadir nuevos periféricos a Algorítmez, la tabla de periféricos en memoria ha cambiado respecto a las especificaciones del simulador software, al reflejar estos nuevos periféricos. Con esto se permite que una rutina de consulta de interrupciones por vector común pueda detectar interrupciones en los cuatro periféricos capaces de generar una interrupción:

Fila	Dir E/S (1)	Estado (1)	Zona E/S (2)	Marco (1)	DF (1)	NBYP (2)	Zona user(2)	Dir aviso(2)	NBL (2)	NBYT (2)	PUNTES (2)	PUNTZU (2)
0	2	0x80	0	0	0	0	0	0	0	0	0	0
1	0	0x80	0	0	0	0	0	0	0	0	0	0
2	4	0x80	0	0	0	0	0	0	0	0	0	0
3	6	0x80	0	0	0	0	0	0	0	0	0	0
4	0xFF	0	0	0	0	0	0	0	0	0	0	0

Las filas se corresponden con los periféricos:

- Monitor. Puerto de estado: 2.
- Teclado. Puerto de estado: 0.
- UART. Puerto de estado: 4.
- Switchs. Puerto de estado: 6.

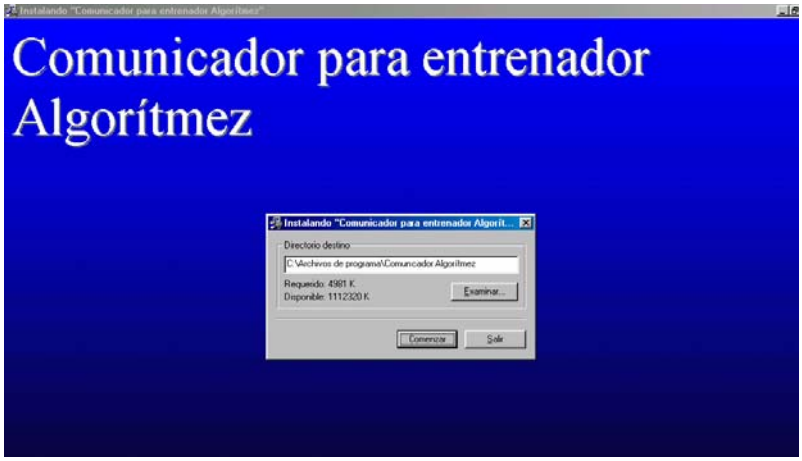
Pantalla:

Por limitaciones del hardware, la pantalla virtual de Algorítmez es una zona de memoria RAM donde se guardan las salidas por el puerto de la pantalla (que se puede visualizar en el LCD o no en cada momento) y contiene solamente 2 filas de 16 caracteres cada una, al igual que el display usado en el entrenador.

A5.4. Comunicador

A5.4.1. Instalación y ejecución

El programa “Comunicador” sólo se encuentra disponible para plataforma Windows. Para instalarlo, hay que descargar el fichero autoinstalable *Comu_Instalar.exe* y al ejecutarlo aparecerá el proceso de instalación:

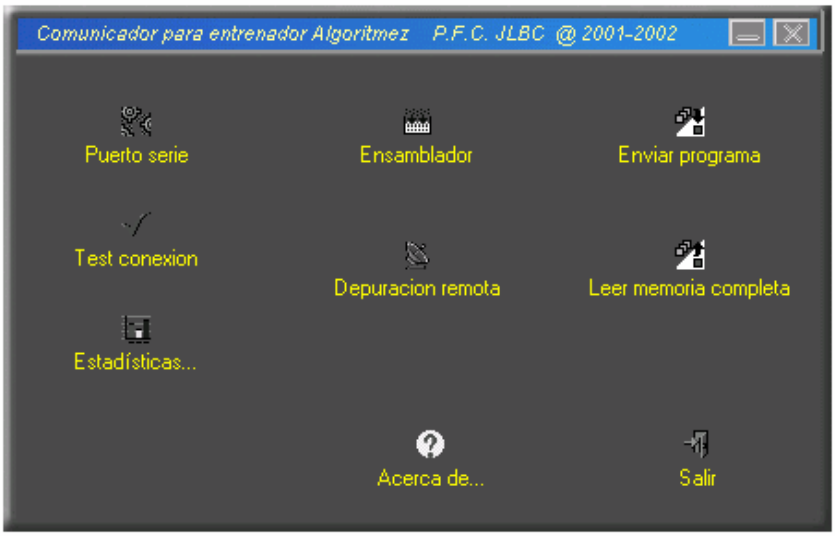


Se elige el directorio donde se quiere instalar el software y pulsando “Comenzar” se copiarán los ficheros necesarios y se crearán los enlaces en el menú Inicio:

Pulsando en el icono “Comunicador de Algorítmez” se lanzará la aplicación.

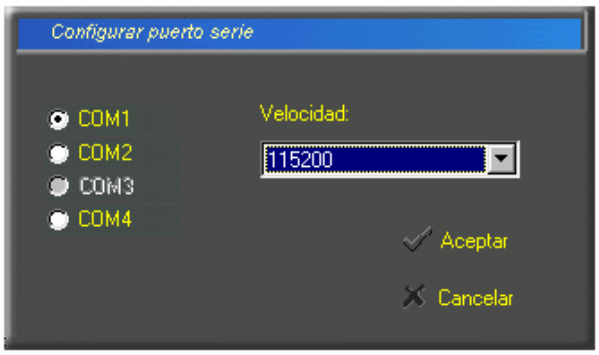
A5.4.2. Ventana principal

La ventana principal de la aplicación consta de una serie de botones que lanzan los distintos módulos:

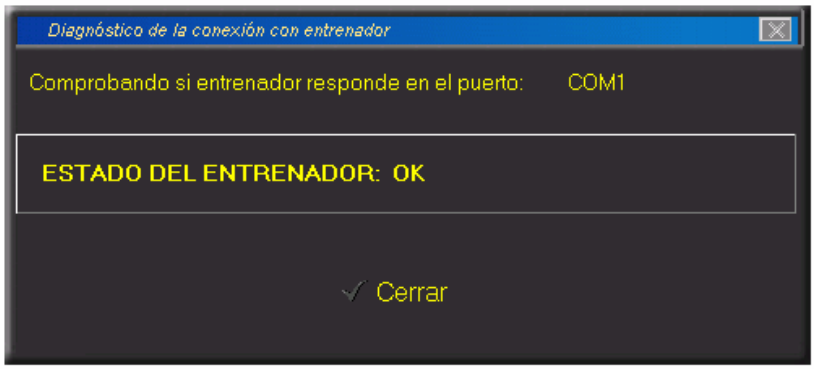


Los módulos correspondientes a los botones “Ensamblador”, “Enviar programa” y “Depuración remota” se explican con más detalle en las siguientes secciones. Del resto se expone a continuación una breve descripción:

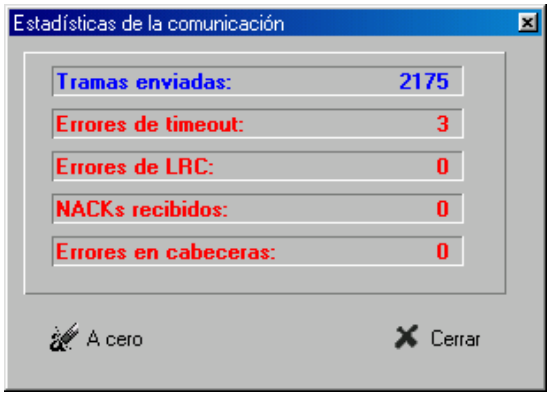
- Puerto serie: Abre una ventana de configuración donde se permite elegir el puerto serie donde se encuentra conectado el entrenador, y la velocidad de datos a usar, aunque para el entrenador esta es siempre de 115200bps a menos que se modifique el firmware.



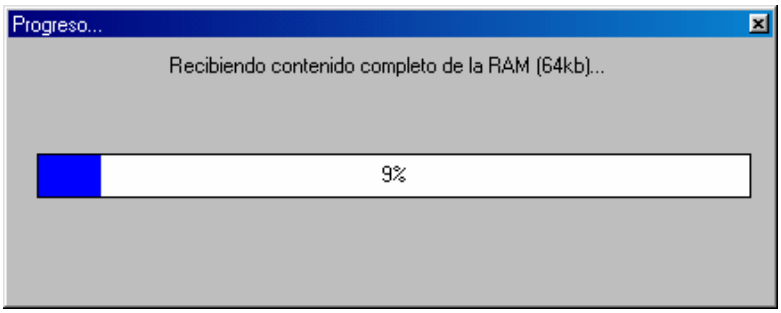
- Test de conexión: Abre una ventana donde se muestra si el entrenador responde correctamente. Esto se realiza enviando cada 3 segundos una trama NOP y verificando la respuesta. Esta opción es útil para verificar la conexión con el entrenador.



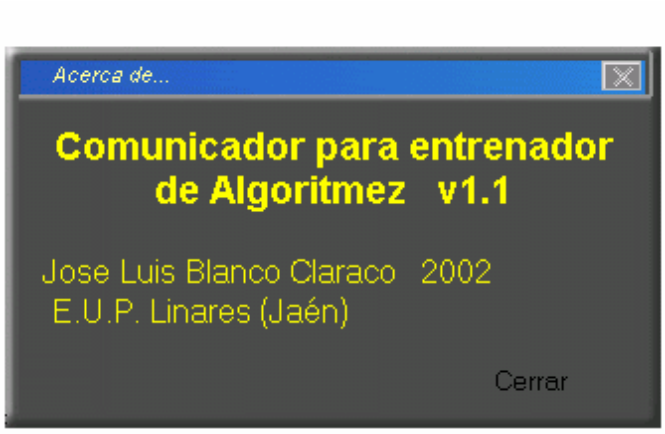
- Estadísticas: Se muestra en una ventana las estadísticas de comunicación con el entrenador, con los siguientes campos:
  - Tramas enviadas: Numero total de tramas enviadas al entrenador.
  - Errores de timeout: Total de ocasiones en que se ha producido este error.
  - Errores de LRC: Contador de fallos detectados en una trama recibida a través de la verificación del campo de LRC.
  - NACKs recibidos: Numero de NACKs recibidos indicando que el entrenador ha detectado un error de LRC.
  - Errores en cabecera: Este fallo indica una desincronización en la recepción de trama.



- Leer memoria completa: Permite el volcado de los 64kbs de memoria RAM emulada de Algorítmez en el momento actual, guardando el resultado en un fichero binario:



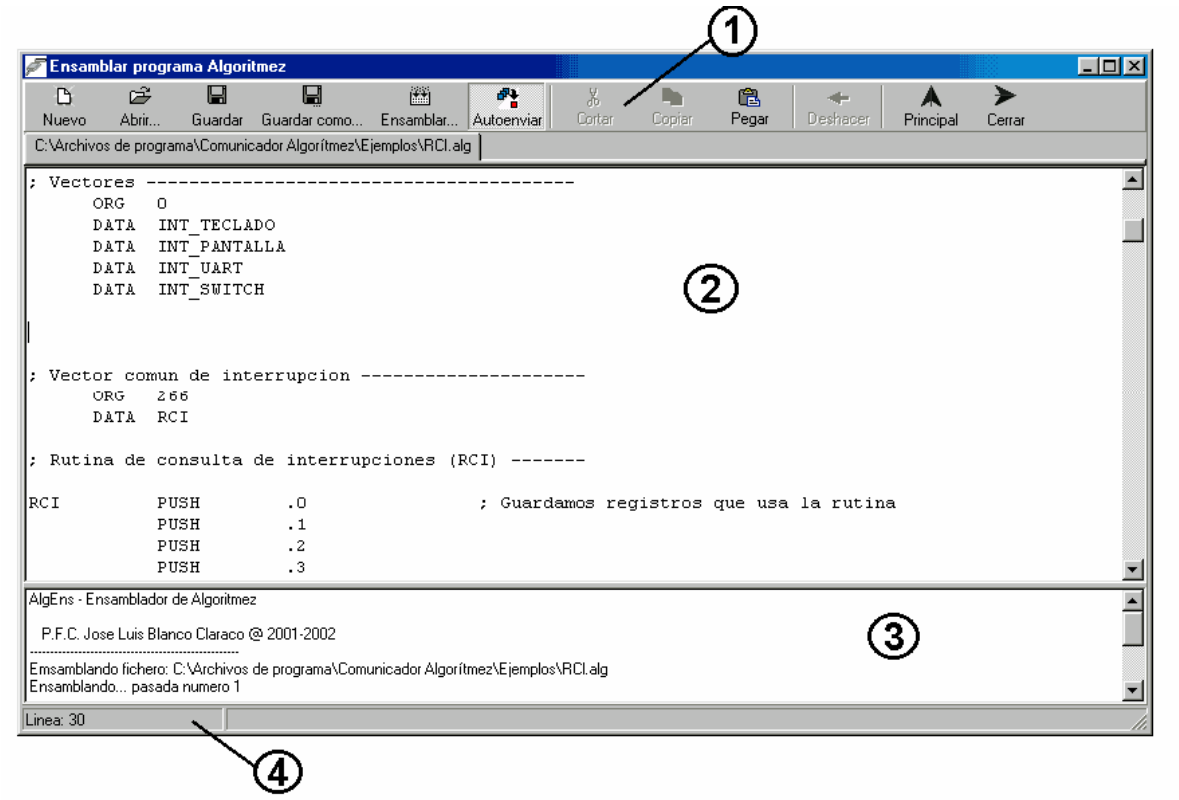
- "Acerca de...": Muestra una ventana con el nombre y versión del programa.



Además, la ventana principal incluye funcionalidad de *tray icon*, de forma que al minimizar ésta, desaparece de la barra de tareas pudiendo restaurarla pulsando sobre su icono:

A5.4.3. Ensamblador

La ventana del módulo editor de programas Algorítmez y ensamblador se compone de las siguientes partes:



1. Barra de botones superior. La componen los siguientes botones:

   - Nuevo. Elimina el contenido del editor dejándolo en blanco.
   - Abrir. Carga un fichero fuente (*.alg).
   - Guardar. Guarda el programa en edición en su correspondiente fichero. Si aún no se ha guardado este programa a disco, se pregunta el nombre que se le quiere dar.
   - Guardar como. Guarda el programa actual con el nombre que se elija.
   - Ensamblar. Realiza un guardado automático del programa, e invoca al programa “AlgEns” para que intente ensamblarlo. Si ocurre algún error estos se muestran en el cuadro inferior.
   - Autoenviar. Este botón cambia su estado marcado/desmarcado al pulsar sobre él, mostrándose hundido si está marcado. Si se activa se abrirá automáticamente la ventana de envío de programas al entrenador con el

fichero objeto generado tras el ensamblado siempre que el programa Algorítmez no contenga ningún error.

- Cortar, copiar, pegar y deshacer: Las operaciones básicas de edición.
- Principal: Muestra la ventana principal de la aplicación, de forma que se pueda entrar en modo simulación, por ejemplo, sin necesidad de cerrar el editor y volver a abrirlo.
- Cerrar: Cierra esta ventana

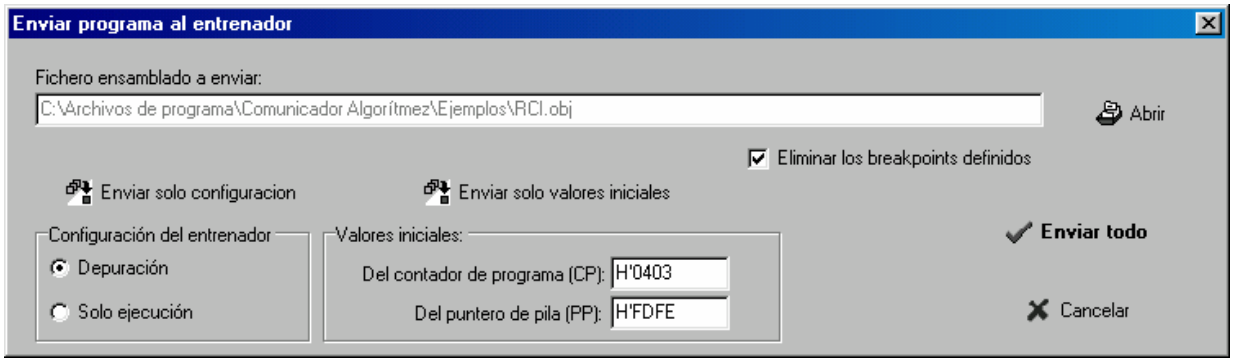
2. Zona de edición. Donde se edita el código fuente del programa Algorítmez.

3. Zona de mensajes del ensamblador. Hacia este cuadro se redirecciona la salida estándar de un proceso del programa "AlgEns" que se encarga del ensamblado del programa. Aquí se listan los errores detectados y haciendo doble clic sobre uno de ellos, se seleccionará la línea del programa al que hace referencia.

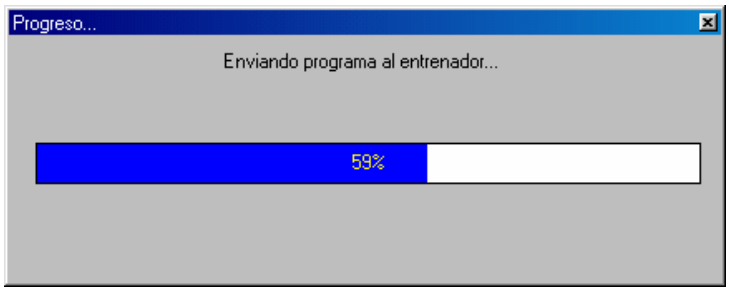
4. Barra de estado. Aquí se muestra el número de línea sobre el que se encuentra el cursor. Además, cuando un programa se ha modificado y no se ha guardado, se mostrará el texto "(MOD)" junto al número de línea.

A5.4.4. Enviar programa

Desde este módulo se permiten las siguiente operaciones:



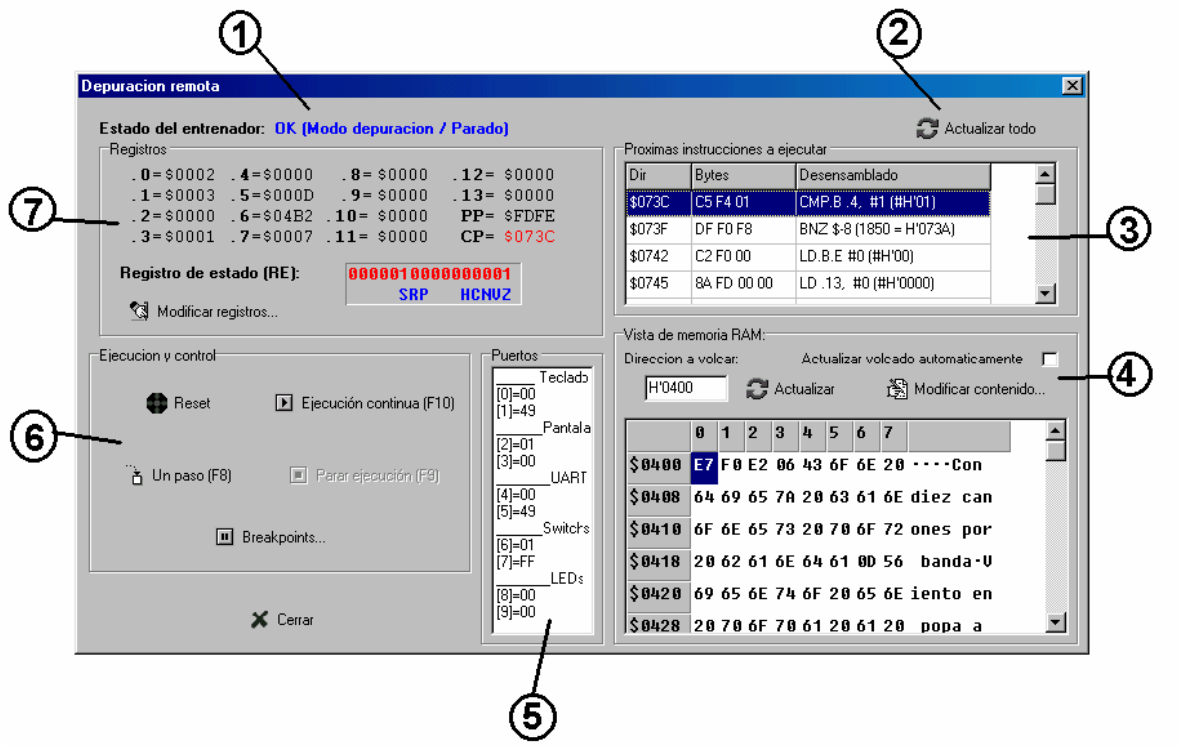
- "Abrir": Permite seleccionar un fichero objeto Algorítmez (*.obj) para el envío al entrenador. Extrae de este fichero los valores iniciales de la pila y del contador de programa y los muestra en el cuadro "Valores iniciales".
- Casilla "Eliminar los breakpoints": Si está señalada, al ejecutar cualquiera de las tres operaciones de envío, se eliminará la lista de breakpoints definida en el entrenador. Si no está marcada, esta lista se mantendrá.
- "Enviar solo configuración": Sólo cambia el modo de funcionamiento del entrenador (según se seleccione en el cuadro "Configuración del entrenador"), sin alterar el programa almacenado. Tras enviar este comando el entrenador se reinicia automáticamente para que la configuración tenga efecto.
- "Enviar solo valores iniciales": Sólo envía los valores que aparecen en el cuadro "Valores iniciales". Tras enviar este comando el entrenador se reinicia automáticamente para que los nuevos valores tengan efecto.
- "Enviar todo": Se envía al entrenador el contenido del nuevo programa a emular, el modo de funcionamiento y los valores iniciales. Tras enviar este comando el entrenador se reinicia automáticamente.



A5.4.5. Depuración remota

Este es el módulo más complejo y útil de la aplicación. Mediante comandos enviados a través del enlace por puerto serie con el entrenador, se obtiene una representación del estado de la máquina virtual Algorítmez en el entrenador, además de permitir modificar ciertos elementos y controlar el flujo de ejecución.

La ventana está dividida en varias zonas, como se ve a continuación:



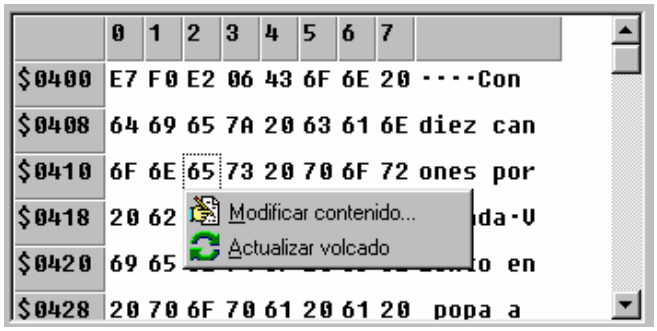
1. Estado del entrenador. Dependiendo del resultado obtenido en la última comunicación, aquí se mostrará:
   - Error: Indica que ha ocurrido un error de comunicación con el entrenador.
   - Ok (<Modo>). La última comunicación se ha realizado sin errores. Además se da el estado actual del entrenador: “Modo ejecución”, “Modo depuración y parado” y “Modo depuración y ejecutando”.

2. “Actualizar todo”. Provoca una sucesión de comandos para actualizar en pantalla el estado real del entrenador. Normalmente no es necesario usar este botón.

3. “Próximas instrucciones”. Mediante una petición de volcado en la zona de ejecución actual y su posterior desensamblado, se muestra un pequeño numero de instrucciones que se ejecutarán a continuación.
4. “Vista de memoria RAM”. Se muestra una parte de la memoria RAM actual. La dirección de comienzo del volcado se establece en “Dirección a volcar”. Debido a que obtener este volcado lleva algunas décimas de segundo (mucho mas que el resto de comandos) por defecto no se actualiza al realizar una actualización general, a menos que se marque la casilla “Actualizar volcado automáticamente”. Además, se puede realizar el volcado de la RAM en cualquier momento pulsando el botón “Actualizar” de dicho cuadro.

   Se puede modificar el contenido de cualquier dirección de RAM pulsando en “Modificar contenido...”, se pedirá la dirección que se quiere modificar y el nuevo contenido y se enviará el comando correspondiente al entrenador para realmente realizar la operación.

   Además para mayor comodidad, pulsando sobre la tabla con el botón derecho, se muestra un menú contextual con estas operaciones:

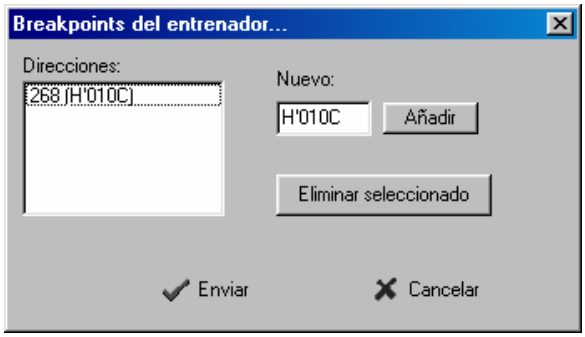


5. “Puertos”. Aquí se muestra el estado actual de los puertos usados en el entrenador.
6. Grupo “Ejecución y control”. Contiene las siguientes operaciones:
   - Reset. Reinicia la máquina virtual Algorítmez en el entrenador, esperando a que éste reconstruya la imagen de la memoria RAM para mostar el estado de los registros, memoria, etc..
   - Un paso. Ejecuta una sola instrucción en el entrenador y actualiza el estado tras ésta.

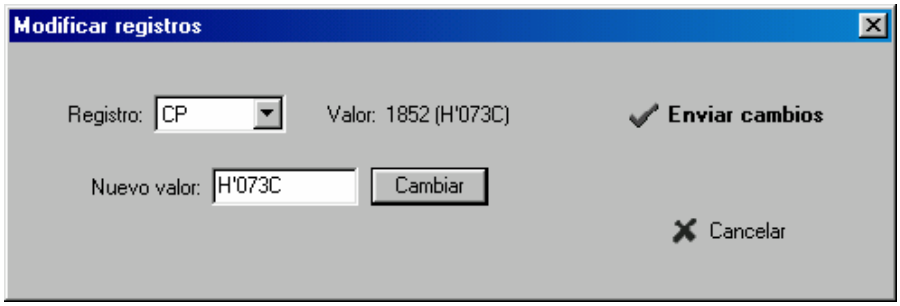
- Ejecución continua. Inicia el modo de ejecución continua de instrucciones en el entrenador. El estado de todos los elementos se recarga cada 2 segundos.
- Parar ejecución. Permite parar el modo de operación continua. Sólo se puede parar un programa cuando el entrenador se ha configurado en modo de depuración (Desde la ventana “Enviar”).

Las teclas de acceso rápido a las funciones de “Un paso”, “Ejecución continua” y “Parar ejecución”, que son F8, F10 y F9 respectivamente, se han elegido coincidiendo con las mismas teclas que realizan la misma función en el entrenador, cuando este está en modo depuración.

- Breakpoints. Abre la siguiente ventana, desde la que se pueden añadir o quitar los hasta 4 puntos de ruptura (breakpoints) que soporta el entrenador. Si la ejecución del programa llega a uno de estos puntos y el entrenador se configuró en modo depuración, se producirá una parada de la ejecución.



7. “Registros”. Se muestra el estado actual de los 16 registros de Algorítmez y el del registro de estado. Se resaltan en rojo los registros o bits del registro de estado cuyos valores hayan cambiado. Pulsando en modificar registros, aparece una ventana donde se pueden modificar los valores que se deseen, y después enviar estos cambios al entrenador:



A5.4.6. Programa “AlgEns”

Este sencillo programa de línea de comando incluido con la aplicación “Comunicador” permite el ensamblado de ficheros fuente escritos en ensamblador para Algorítmez, produciendo ficheros objeto con el siguiente formato:

Campo	**Offset (bytes)**	**Longitud (bytes)**	**Descripción**
Magic	0x00000	0x00004	Es constante = 0x44776172 Asegura que es un fichero del formato adecuado.
Contenido	0x00004	0x10000	La imagen de los 64kbytes de memoria de Algorítmez con el programa ensamblado.
Valor inicial de CP	0x10004	0x00002	Depende del programa.
Valor inicial de PP	0x10006	0x00006	Siempre a 0xFDE
TAMAÑO TOTAL		**0x10008**	

Hay que señalar que los valores iniciales de CP y PP están almacenados con el byte alto primero (al estar hecho en Java y ser este el orden de los bytes), lo que hay que tener en cuenta si se carga el fichero desde un programa en otro lenguaje de programación como C++ donde el orden es el inverso.

A continuación se muestra un ejemplo de la ejecución de este programa:

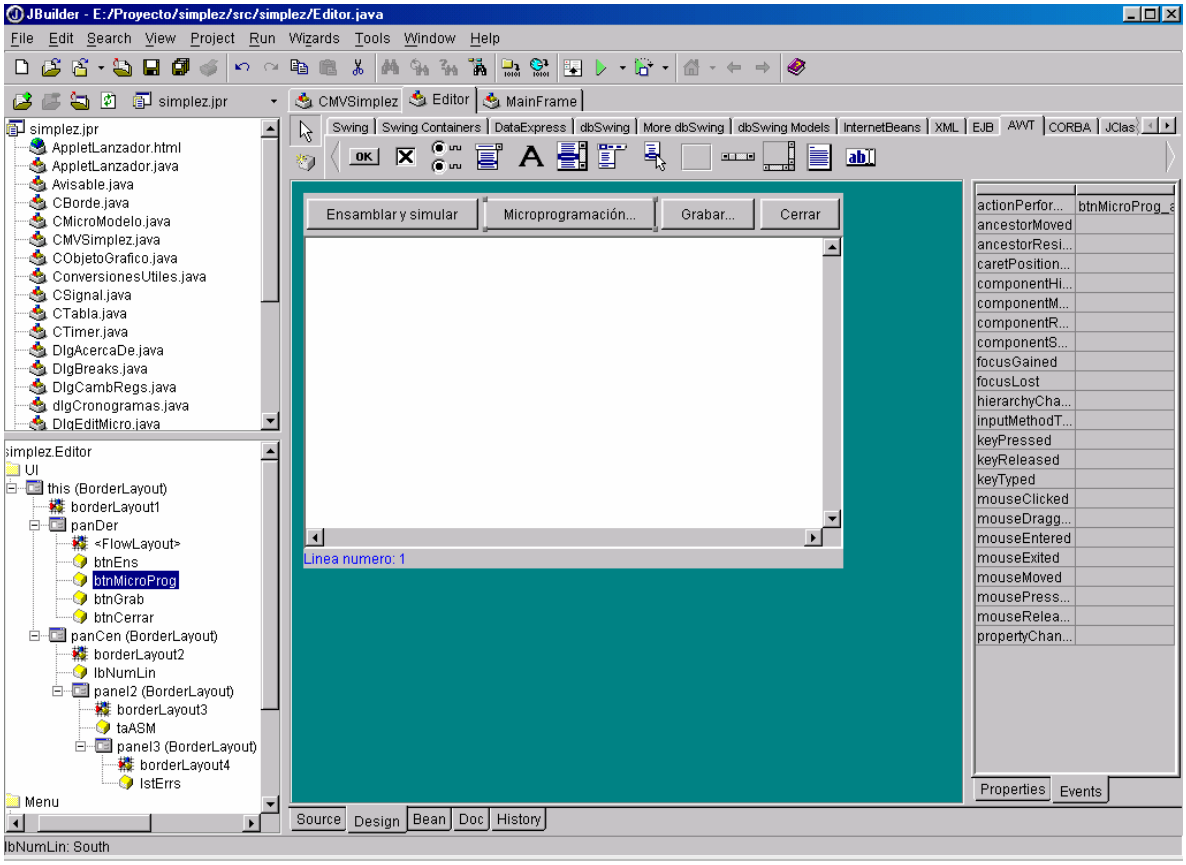
```
C:\Archivos de programa\Comunicador Algorítmez>algens ejemplos/RCI.alg
AlgEns - Ensamblador de Algoritmez
   P.F.C. Jose Luis Blanco Claraco @ 2001-2002
------------------------------------------------
Emsamblando fichero: ejemplos/RCI.alg
Ensamblando... pasada numero 1
Ensamblando... pasada numero 2
Ensamblado finalizado SIN errores
     Se han encontrado 4 simbolos
     Se han encontrado 24 etiquetas
Se ha creado fichero objeto: ejemplos/RCI.obj

C:\Archivos de programa\Comunicador Algorítmez>_
```

A6. Manuales de mantenimiento de software

A6.1. Simulador de Símplez

A6.1.1. Introducción

Para este simulador se ha usado el entorno de desarrollo JBuilder 6 de Borland. Este entorno incorpora además de un editor de código y una representación en árbol de las clases en tiempo real, un editor visual de interfaces de usuario:



Hay que señalar que las interfaces gráficas creadas con JBuilder se basan completamente en código Java estándar, guardando toda la información gráfica en los mismos ficheros de código fuente. Para que sea posible activar el editor visual sobre una clase debe ser del tipo Applet, Dialog, Frame o Panel.

Además de diseñar las interfaces gráficas y colocar los componentes en estas, se pueden colocar procesadores para los eventos de cada componente, lo que se ve en el inspector de objetos, a la derecha en la anterior imagen.
Al haber decidido no usar Java 2 en estos desarrollos se han usado solamente los componentes gráficos del paquete AWT, en lugar de los más actuales componentes Swing.

A6.1.2. Lista de clases Java

Todas las clases e interfaces usadas en el simulador de Símplez se han creado en un *package* llamado "simplez". Además se ha usado el *package* común "componentes" que ya se ha descrito en el anexo A2. A continuación se listan las clases e interfaces que forman el *package* "simplez" junto a una pequeña explicación de su uso:

Interfaces:

- InterfaceVMRepaint

Clases:

- AppletLanzador. Hereda de Applet. Implementa al applet que se llama desde los navegadores. Este applet no realiza nada más que crear una clase MainFrame, lo que hará que aparezca la ventana principal de la aplicación. También se incluye un método *main* para la ejecución desde fuera de navegadores. Este método también realiza solamente la misma tarea.
- CMicroModelo. Hereda de Object. Esta clase es usada para representar a la configuración de la micromáquina Símplez. Esto incluye la memoria de control, los mnemónicos asociados a cada opcode y un indicador de uso de operando para cada uno de estos opcodes.

- CMVSimplez. Hereda de Object. Esta es la clase principal de la aplicación. Incluye una representación de una máquina virtual Símplez y permite la emulación de todas sus microseñales, además del ensamblado de programas.
- CObjetoGrafico. Hereda de Object. Usada para almacenar un objeto gráfico específico: Textos fijos, líneas, cuadros, etc.. Es usado para renderizar la imagen de la micromáquina de Símplez dinámicamente, dependiendo del estado de las distintas señales. En su constructor se usa una cadena de valores separados por comas, que se separan y se guardan en un vector interno. El primer valor identifica el tipo de gráfico, siendo los demás valores coordenadas, colores, etc...
- CSignal: Hereda de Object. Se usa como estructura de datos para representar a una señal, bus o registro de la micromáquina. Se incluyen los campos: Nombre de la señal, longitud en bits y valor actual. Además se define una constante especial: HI_Z que asignada a una señal de tipo bus, indica que ese bus se encuentra en estado de alta impedancia.
- DlgAcercaDe. Hereda de Dialog. Implementa el cuadro de diálogo “Acerca de..”.
- DlgBreaks. Hereda de Dialog. Implementa el diálogo de edición de breakpoints. Se le pasa una referencia a la clase CMVSimplez usada por el simulador, para editar directamente su lista de breakpoints.
- DlgCambRegs. Hereda de Dialog. Implementa el cuadro de diálogo para cambiar el valor del acumulador y del contador de programa.
- DlgCronogramas. Hereda de Dialog. Implementa la ventana de visión de cronogramas. Con cada semiciclo que se ejecuta en la micromáquina de Símplez, se guarda en un vector un vector con todos los valores de las señales en cada instante. De esta forma es posible seleccionar posteriormente las señales que se quieren visualizar.
- DlgEditMicro. Hereda de Dialog. Implementa la ventana de edición de la memoria de control para la micromáquina. Incluye opciones de grabación y carga de modelos.
- DlgEtiquetas. Hereda de Dialog. Es el cuadro de diálogo donde se muestran los valores de las etiquetas encontradas durante el ensamblado. Se llama desde el módulo principal del simulador.

- DlgMicroSimplez. Hereda de Dialog. Implementa la ventana que muestra gráficamente la micromáquina, usando un vector de objetos CObjetoGrafico.
- DlgSimulador. Hereda de Dialog. Es la ventana principal del simulador, donde se encuentran las tablas de memoria y los botones de control de la simulación. Uno de sus miembros es un objeto de la clase CMVSimplez, que se usa para realizar toda la simulación.
- Editor. Hereda de Frame. Es el editor de programas en ensamblador. Usa un objeto de la clase CMVSimplez para realizar el ensamblado y si no se producen errores, pasa una referencia a este mismo objeto al diálogo DlgSimulador.
- HiloEjecutor. Hereda de Thread. Es un hilo que se usa para el modo de ejecución continuo. En su ejecución se alternan periodos de inactividad (necesario para no bloquear toda la aplicación) con periodos donde se ejecutan un gran número de instrucciones en el simulador, actualizando a continuación la ventana del simulador, de la que se le ha pasa una referencia.
- MainFrame. Hereda de Frame. Implementa la ventana principal, con un menú desde el que se accede al editor de programas, a la ayuda y al diálogo “Acerca de”.

A6.2. Simulador de Algorítmez

A6.2.1. Introducción

Este simulador también se ha realizado en Java y usando el entorno de desarrollo JBuilder 6 de Borland, por lo que se aplica lo mismo que se explica en A6.1.1. referente al simulador de Símplez.

A6.2.2. Lista de clases Java

Todas las clases e interfaces usadas para el simulador Algorítmez se han encapsulado en un *package* llamado "algoritmez", usando además el *package* común "componentes" que ya se ha descrito en el anexo A2. A continuación se listan las clases e interfaces que forman el *package* "algoritmez" junto a una pequeña explicación de su uso:

Interfaces:

- InterfaceVMRepaint: Esta interfaz solo contiene un método sin parámetros "`void OnVMRepaint()`". Esta interfaz se usa para implementarla en el cuadro de diálogo del simulador, pasando una referencia a esta al constructor de la clase CMVAlgoritmez, que usara esta interfaz para actualizar en pantalla el estado de la memoria, etc... cuando sea necesario.
- InterfazPantalla. Usada para el manejo de la pantalla de Algorítmez, en la ventana "Consola de E/S". Define los siguientes métodos:

```
interface InterfazPantalla
{
  char      getContenidoPantalla(int col,int row);
  void      setContenidoPantalla(int col,int row,char c);
  int       getCursorX();
  int       getCursorY();
  void      setCursorX(int x);
  void      setCursorY(int y);
  boolean   getListo();
  void      setOcupado();
```

```
  void      ActualizaPantalla();
}
```

Como se puede observar define un interfaz con el que se puede controlar completamente el contenido de la pantalla, de forma que se consigue independencia de la clase principal CMVAlgoritmez con el resto de diálogos y ventanas, lo que da más portabilidad.

Clases:

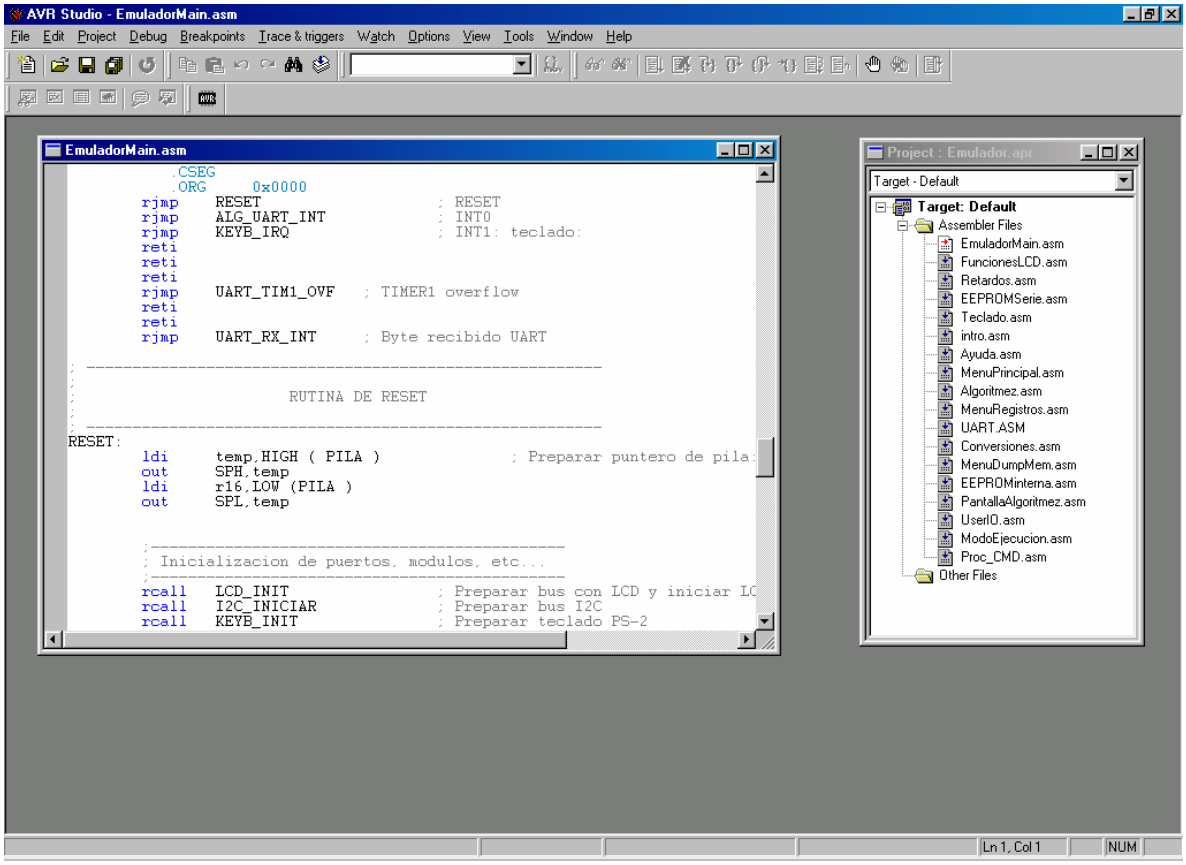
- AppletLanzador. Hereda de Applet. Implementa al applet que se llama desde los navegadores. Este applet no realiza nada más que crear una clase MainFrame, lo que hará que aparezca la ventana principal de la aplicación. También se incluye un método *main* para la ejecución desde fuera de navegadores. Este método también realiza solamente la misma tarea.
- CMVAlgoritmez. Hereda de Object. Es la clase principal de la aplicación ya que implementa el ensamblador y el simulador de Algorítmez.
- DlgAcercaDe. Hereda de Dialog. Ventana “Acerca de..”.
- DlgBreaks. Hereda de Dialog. Ventana que muestra una lista de los breakpoints establecidos en el simulador de Algorítmez. Escribe y lee directamente en el vector “Breakpoints” del objeto CMVAlgoritmez usado para la simulación.
- DlgConsola. Hereda de Dialog. Implementa la interfaz “InterfazPantalla” para mostrar en una ventana el contenido de la pantalla de Algorítmez en un Canvas. Además incluye un TextField para el teclado y un Label para representar la tecla en buffer actual.
- DlgEtiquetas. Hereda de Dialog. Es la ventana donde se muestran las etiquetas y símbolos encontrados durante el ensamblado.
- DlgRegistros. Hereda de Dialog. Una ventana desde la que se puede modificar el contenido de los registros y el registro de estado de Algorítmez. Se le pasa una referencia al objeto de la clase CMVAlgoritmez que se usa en el simulador.
- DlgSimulador. Hereda de Dialog. Es la ventana principal del simulador. Implementa la interfaz “InterfaceVMRepaint”. Usa un objeto de la clase CMVAlgoritmez que le pasa desde Editor para la simulación.

- Editor. Hereda de Frame. Es la ventana de edición de programas fuente de Algorítmez. Instancia un objeto de clase CMVAlgoritmez para realizar el ensamblado y si no ocurren errores, abrir una ventana de la clase "DlgSimulador".
- EnsStringTokenizer. Hereda de Object. Se ha implementado este tokenizer que permite separar cadenas separadas por comas, pero respetando cadenas de caracteres encerradas entre comillas dobles. Se usa durante el análisis de las pseudoinstrucciones DATA y DATA.B
- HiloEjecutor. Hereda de Thread. Es un hilo que se usa para el modo de ejecución continuo. En su ejecución se alternan periodos de inactividad (necesario para no bloquear toda la aplicación) con periodos donde se ejecutan un cierto número de instrucciones en el simulador, actualizando a continuación la ventana del simulador, de la que se le ha pasa una referencia.
- MainFrame. Hereda de Frame. Implementa la ventana principal, con un menú desde el que se accede al editor de programas, a la ayuda y al diálogo "Acerca de".

A6.3. Firmware del entrenador

A6.3.1. Introducción

El firmware para el microcontrolador central del entrenador, el AT90S8515 de la familia AVR, se ha escrito en ensamblador para aprovechar al máximo tanto el espacio de código como la velocidad de proceso. Tanto para el ensamblado de los ficheros fuentes como para su simulación en el PC (durante las fases de depuración iniciales) se ha usado el entorno gratuito proporcionado por el fabricante en www.atmel.com, el AVR Studio 3.5:



Este entorno es un sencillo IDE en el que se define un fichero de proyecto que contiene una lista de ficheros fuentes. Solo uno de estos se marca como punto de entrada al ensamblador, estando el resto de módulos incluidos mediante directivas “include” en el módulo principal.

A6.3.1. Módulos en ensamblador

A continuación se da un listado de los módulos que se han escrito y una breve descripción de las funciones principales que realizan sus rutinas:

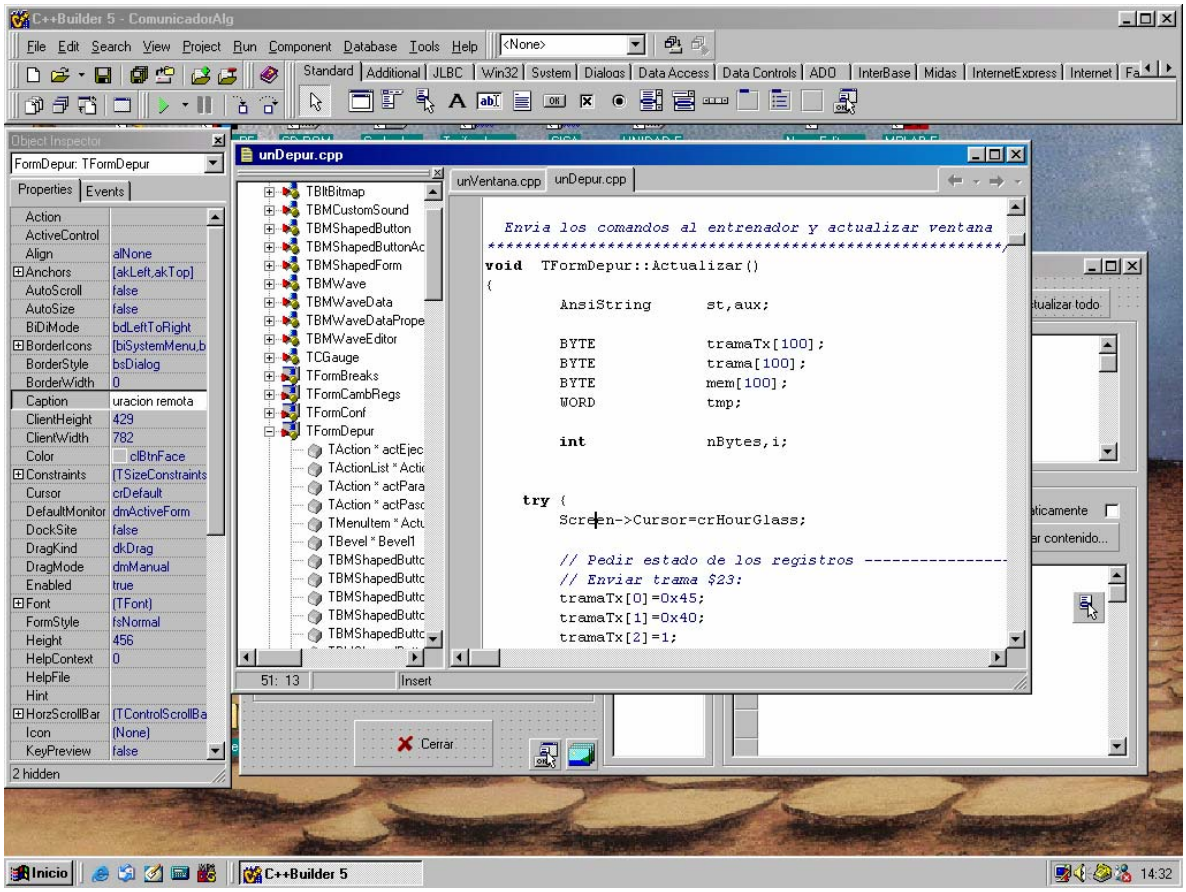
- EmuladorMain. Es el módulo principal del programa. Desde aquí se importan el resto de módulos mediante directivas “include”. En este módulo se implementa además la tabla de vectores de interrupciones y se procesa el vector de reset, donde se inician todos los periféricos, se carga la configuración del entrenador de la eeprom interna y se pasa a modo de solo ejecución o de depuración según el caso.
- FuncionesLCD. Implementa una gran cantidad de rutinas para el manejo de la pantalla LCD. Además se implementan algunas sencillas rutinas de conversión de binario a cadena de binario, a cadena hexadecimal y a cadena en decimal.
- Retardos. Aquí se implementan tres rutinas de retardo cuyo retardo es función lineal de un valor pasado en el registro r25, y otra rutina que realiza el retardo de bit de la UART emulada de Algorítmez. Para esta ultima se usa un temporizador hardware del microcontrolador, el TIMER0.
- EEPROMSerie. Aquí se implementan las rutinas base de control de bus I2C y otras rutinas de más alto nivel que, usando las rutinas I2C, permiten la lectura y escritura en los bancos de memoria externa, tanto de un solo byte, como de varios bytes, usando los modos de acceso secuenciales de las memorias.
- Teclado. En este módulo se implementa la rutina procesadora de interrupciones del teclado, que recibe y decodifica los correspondientes scan codes, usando una tabla para obtener los correspondientes códigos ASCII. Mediante una variable en RAM se indica que hay una tecla en el buffer de entrada.
- Intro. Este módulo muestra por pantalla el mensaje inicial en el display tras reiniciar.
- Ayuda. Aquí se implementa la ayuda que se muestra al pulsar F1 en la pantalla principal.
- MenuPrincipal. Es la pantalla principal del modo de depuración, donde se muestra la siguiente instrucción a ejecutar desensamblada. Desde aquí se comprueban las teclas pulsadas para ir al resto de los menús.

- Algoritmez. En este módulo es donde se realiza todas las tareas de simulación de la máquina virtual Algorítmez, así como algunas tareas auxiliares, como el desensamblado de instrucciones o cálculo de número de bytes usados por cada una, datos usados por el menú principal para mostrar la instrucción actual.
- MenuRegistros. Aquí se implementan los menús de visor de registros y elección de registro a editar.
- UART. Este módulo contiene la rutina que procesa las interrupciones de la UART del microcontrolador. Aquí es donde se implantan las funciones asignadas a la capa de enlace del protocolo de comunicaciones. Mediante un indicador en RAM se indica que una trama esta completamente recibida para su proceso por la capa de comandos.
- Conversiones. Aquí se implementan algunas rutinas útiles como conversión de números a hexadecimal o decimal. Además se ha implementado una rutina que realiza un salto a una dirección de programa almacenada en una tabla según un valor de clave a buscar en la tabla. Esto es equivalente a una función "switch...case" en un lenguaje de alto nivel.
- MenuDumpMem. Donde se implementa el menú visor de memoria RAM.
- EEPROMInterna. Aquí se incluyen rutinas de lectura y grabación de bloques de datos a la eeprom interna del microcontrolador.
- PantallaAlgoritmez. Se implementa el visor de estado actual de pantalla virtual Algorítmez, correspondiente a la tecla F5 desde la pantalla principal.
- UserIO. Sólo incluye una rutina ("USER_IO_INPUT"), que muestra un texto en la primera línea del display, un texto por defecto en la segunda y permite editar este segundo campo, devolviendo el texto cuando se pulsa INTRO en un buffer en RAM.
- ModoEjecucion. Aquí se implementa la rutina que procesa el modo de solo ejecución del entrenador.
- Proc_cmd. Este modulo implementa la ejecución de los comandos (capa 3 del protocolo de comunicaciones) recibidos del PC.

A6.4. Aplicación “Comunicador”

A6.4.1. Introducción

Esta aplicación se ha desarrollado en lenguaje C++ con la herramienta de Borlad C++ Builder 5:



Los proyectos en este entorno se componen de una serie de ficheros fuente C++ (*.cpp), llamadas unidades, cada una con su correspondiente fichero cabecera (*.h). Algunas de estas unidades se corresponden con formularios (forms) aunque se pueden usar unidades sólo de clases o funciones. Los formularios son clases C++ que encapsulan las funcionalidades de una ventana y donde se pueden colocar los componentes de interfaz de usuario mediante un editor visual.

A6.4.2. Formularios

Esta es la lista de todos los formularios usados en el proyecto junto con sus funciones principales:

- unAboutBox.cpp: Diálogo “Acerca de...”
- unBreaks.cpp: Ventana de edición de lista de breakpoints del entrenador.
- unCambRegs.cpp: Ventana de edición de valor de los registros Algorítmez.
- unConf.cpp: Ventana de configuración para el puerto serie.
- unDepur.cpp: Este es el módulo principal del depurador remoto.
- unDiagnos.cpp: Cuadro de diálogo de diagnóstico del enlace con el entrenador.
- unEnsamblar.cpp: Editor de código fuente para programas Algorítmez y creación de procesos de “AlgEns” para el ensamblado de estos.
- unEnviar.cpp: Ventana de envió de programas y de configuración.
- unEstadisticas.cpp: Ventana que muestra las estadísticas de las comunicaciones.
- unModMem.cpp: Edición del contenido de una dirección de memoria RAM.
- unProgreso.cpp: Esta ventana se llama en cada ocasión que se realiza un proceso largo, mostrando una barra de progreso.
- unVentana.cpp: La ventana principal de la aplicación y donde se encuentra el componente para la funcionalidad del *tray icon*.

A6.4.3. Lista de módulos en C++

Como se ha dicho no todas las unidades deben contener una ventana o formulario. A continuación se detalla la lista de las unidades que no son de formularios en el proyecto:

- Algoritmez.cpp: Incluye algunas funciones como ensamblar programas (a través del proceso externo AlgEns), desensamblar una instrucción y evaluación de constantes en formato Algorítmez, como por ejemplo, constantes en hexadecimal con prefijo H’.
- unComs.cpp: Que implementa las funciones de comunicaciones. Existen funciones específicas para casi todos los comandos, como por ejemplo, grabar o leer un bloque de memoria RAM en el entrenador, aunque todas estas usan una función base que es “TxRxTrama” donde se implementan las especificaciones de envío y recepción de tramas en la capa de enlace, incluyendo las retransmisiones y los reintentos.

A7. Hojas de características

En las siguientes páginas se adjuntan las hojas de características (*datasheets*) de los siguientes componentes usados en el entrenador de Algorítmez:

- AT24C512. Memoria EEPROM serie con bus I2C de Atmel
- AT90S8515. Microcontrolador con núcleo RISC de Atmel. Solo la especificaciones resumidas.
- ST232. Convertidor de señales TTL a RS232 y viceversa, de Siemens.