La máquina virtual ROSSI
(Register-Oriented machine with a Set of Simple Instructions)

Antonio Vilar Sánchez

Agosto 2004

1. Introducción

La máquina virtual **ROSSI** (Register-Oriented machine with a Set of Simple Instructions) es una máquina orientada a registros con arquitectura de carga y almacenamiento, similar a la de la mayoría de procesadores actuales. En este tipo de máquinas, las instrucciones de cálculo requieren que sus operandos se encuentren almacenados en registros, y la memoria es accedida únicamente por las instrucciones de carga y almacenamiento.

La máquina ofrece un conjunto de instrucciones simples y proporciona un número de modos de direccionamiento reducido. Además, se trata de una máquina con un diseño no ortogonal, ya que no permite emplear cualquier modo de direccionamiento para cualquier instrucción.

La máquina **ROSSI** dispone de dos bancos de registros de propósito general de tamaño ilimitado[†] (uno para cada tipo de datos con el que opera la máquina). Además, proporciona una serie de registros específicos enteros y reales, destinados a la gestión de la pila, subrutinas, paso de parámetros a las llamadas al sistema, etcétera.

Para simplificar la gestión de la memoria, **ROSSI** dispone de memorias de datos e instrucciones independientes, también de tamaño ilimitado[†]. La memoria de instrucciones es la encargada de almacenar el programa y no puede ser modificada durante la ejecución del mismo. La memoria de datos, por otro lado, contiene la información almacenada por el usuario y se modifica durante la ejecución del programa. Cada posición de la memoria de datos puede contener tanto un valor entero (posiblemente correspondiente al código ASCII de un carácter) como un valor real. Las direcciones de estas memorias se especifican mediante enteros no negativos y ambas comienzan en la dirección 0.

Además, la máquina **ROSSI** dispone de un terminal de entrada/salida que le sirve de interfaz con el mundo exterior. La salida se lleva a cabo mediante la pantalla y la entrada, mediante el teclado.

2. Registros

Como se ha comentado en la introducción, **ROSSI** dispone de dos bancos de registros de tamaño ilimitado: uno destinado a almacenar valores enteros y otro, para los valores reales.

[†] En realidad el tamaño estará limitado por la cantidad de memoria de que disponga la máquina sobre la cual se emule.

Los registros enteros de propósito general se nombran con el carácter dólar seguido del carácter erre minúscula y una secuencia no vacía de dígitos, sin ningún prefijo de ceros, siendo el primero de ellos el registro `$r0`.

Además, existe una serie de registros enteros de propósito específico, cuya funcionalidad se describe en la siguiente tabla:

Registro	Función
`$zero`	Mantiene siempre el valor constante entero 0.
`$pc`	Contiene la dirección de la siguiente instrucción a ejecutar. No es directamente accesible para el usuario.
`$sp`	Contiene la dirección de la primera posición libre de la pila
`$fp`	Puntero de bloque de activación de la subrutina.
`$ra`	Contiene la dirección de retorno en una llamada a subrutina.
`$sc`	Contiene el código de la llamada al sistema.
`$a0` `$a1`	Utilizados para el paso de parámetros y devolución de resultados de tipo entero en una llamada al sistema.

Tabla 1. Registros enteros de propósito específico en una máquina **ROSSI**.

Los registros reales de propósito general se nombran con el carácter dólar seguido del carácter efe minúscula y una secuencia no vacía de dígitos, sin ningún prefijo de ceros, siendo el primero de ellos el registro `$f0`.

Al igual que sucede en el caso entero, la máquina ofrece una serie de registros reales de propósito específico, tal y como se describe en la siguiente tabla:

Registro	Función
`$fzero`	Mantiene siempre el valor constante real 0.0.
`$fa`	Utilizado para el paso de parámetros y devolución de resultados de tipo real en una llamada al sistema.

Tabla 2. Registros reales de propósito específico en una máquina **ROSSI**.

En la máquina **ROSSI**, cualquier acceso de lectura sobre un registro no inicializado previamente provoca una excepción, dejando la máquina detenida en un estado de error (excepto en las instrucciones `save` y `fsave`, como se comentará más adelante).

3. Modos de direccionamiento

Para especificar la ubicación de los operandos de una instrucción, la máquina **ROSSI** proporciona los cuatro modos de direccionamiento siguientes:

Nombre	Formato
Directo a registro	*$registro*
Indirecto a registro con desplazamiento	*entero($registro)*
Inmediato	*entero* o *real*
Etiqueta	*etiqueta*

Tabla 3. Modos de direccionamiento proporcionados por la máquina **ROSSI**.

- **Directo a registro:** el operando se encuentra almacenado en el registro *$registro*. Este modo de direccionamiento se utiliza en todas las instrucciones (excepto en las instrucciones `j`, `jal` y `syscall`).

  Por ejemplo, la instrucción `add $r3, $r1, $r2` suma los enteros almacenados en los registros `$r1` y `$r2` y almacena el resultado en el registro `$r3`.

- **Indirecto a registro con desplazamiento:** el operando se encuentra almacenado en la posición de memoria resultante de sumar el desplazamiento *entero* al contenido del registro entero *$registro*. El desplazamiento es un número entero que se ajusta a la siguiente expresión regular: `[+-]?[0-9]+`. Este modo de direccionamiento es el utilizado para especificar la dirección de memoria en la que se encuentra el operando en las instrucciones de carga y almacenamiento `sw`, `save`, `lw`, `rest`, `fsw`, `fsave`, `flw` y `frest`.

  Por ejemplo, si el registro `$r15` contiene el entero 12 y la posición de memoria 16 contiene el real 1.5, la ejecución de la instrucción `flw $f0, 4($r15)` almacena el valor 1.5 en el registro real `$f0`.

- **Inmediato:** el operando se encuentra de modo explícito en la propia instrucción, siendo éste un valor entero o real. La expresión regular a la que se ajusta *entero* es la misma que en el caso anterior. La expresión regular asociada a *real* es la siguiente: `[+-]?[0-9]+\.[0-9]+([eE][-+]?[0-9]+)?`.

  Las instrucciones que admiten este modo de direccionamiento se diferencian de las demás por tener un código de instrucción finalizado con el carácter i minúscula. Por ejemplo, la instrucción `faddi $fa, $fzero, -1.25e2` almacena el valor real -125.0 en el registro `$fa`.

- **Etiqueta:** el operando es la línea en la que se define *etiqueta*. El formato que debe seguir `etiqueta` se explica con detalle en el siguiente apartado. Este modo de direccionamiento es utilizado por la mayoría de instrucciones de salto.

  Por ejemplo, si la etiqueta `bucle` está definida en la línea 10, la ejecución de la instrucción `j bucle` almacena el valor 10 en el registro `$pc`.

  Cabe mencionar aquí que el uso de una etiqueta que no ha sido definida a lo largo del programa dará lugar a un error semántico, por lo que el programa resultante no será correcto.

4. Estructura de los programas

Un programa **ROSSI** está formado por una secuencia no vacía de líneas que cumple las características que se mencionan a continuación.

Los comentarios, espacios, tabuladores y líneas en blanco pueden aparecer en cualquier parte del programa y son ignorados por la máquina virtual. Los comentarios comienzan con el carácter almohadilla (#) y abarcan desde éste hasta el final de la línea.

En un programa **ROSSI** la caja es significativa, esto es, se distingue entre mayúsculas y minúsculas, por lo que identificadores como `et1` y `ET1` no referencian a la misma etiqueta.

Inicialmente puede aparecer, de modo opcional, la directiva `.data`, que indica el comienzo de la zona de inicialización de las cadenas del programa. En esta zona, que de existir, no puede ser vacía, se establece el valor de las cadenas en la memoria de datos, haciendo uso de la directiva `.asciiz`. La sintaxis de la directiva `.asciiz` es la siguiente:

`.asciiz` *`entero literal_cadena`*

donde:

- *`entero`* es opcional y es un entero no negativo (con la misma expresión regular que en los casos anteriores) que, en caso de aparecer, especifica la posición de memoria a partir de la cual se debe almacenar la cadena. Si no se especifica ningún valor para *`entero`*, la cadena se empieza a almacenar a partir de la siguiente posición de memoria libre a continuación de la última cadena almacenada (o a partir de la posición 0 si es la primera cadena que se inicializa).

- *`literal_cadena`* es un literal de cadena que viene delimitado por comillas dobles (") y que no podrá contener en su interior comillas dobles, saltos de línea, tabuladores, ni el carácter barra invertida (\). Para permitir la aparición de estos caracteres en su interior se deben introducir como las secuencias de escape \", \n, \t y \\, respectivamente, siendo éstas las únicas secuencias de escape permitidas.

El resultado de ejecutar una directiva `.asciiz` consiste en almacenar en la memoria de datos, de modo consecutivo, el código ASCII de cada uno de los caracteres que forman el literal de cadena, indicando el fin del mismo mediante el valor entero 0 (carácter nulo). Así, para la directiva `.asciiz 3 " a1"`, el contenido de las posiciones de memoria 3, 4, 5 y 6 será 32, 97, 49 y 0, respectivamente. Si a continuación se ejecuta la directiva `.asciiz "A1\n"`, ésta almacenará en las posiciones de memoria 7, 8, 9 y 10 los valores 65, 49, 10 y 0, respectivamente.

La directiva `.text`, que debe aparecer obligatoriamente en todos los programas, marca el final de la zona de inicialización de cadenas (en caso de que la hubiera) y el inicio de la zona de instrucciones, indicando además que las instrucciones que la siguen se deben almacenar en la memoria de instrucciones.

La zona de instrucciones debe contener al menos una instrucción, y está formada por una secuencia no vacía de líneas de la forma:

`defetiqueta instruccion`

donde:

- *`defetiqueta`* es opcional y representa una definición de etiqueta, consistente en una secuencia de caracteres alfanuméricos (posiblemente correspondiente a una palabra reservada), que no empieza con un dígito, y que finaliza con el carácter dos puntos (`:`). La definición de una etiqueta que ya ha sido definida previamente producirá un error semántico, dando lugar a un programa incorrecto.

- *`instruccion`* también es opcional y es una de las 47 instrucciones que ofrece la máquina.

Las instrucciones proporcionadas por la máquina **ROSSI** se encuentran detalladas en el siguiente apartado.

5. Juego de instrucciones

A continuación se muestra detallado el juego completo de instrucciones que proporciona la máquina **ROSSI**.

5.1. Instrucciones aritmético-lógicas

Las siguientes instrucciones exigen que sus operandos sean de tipo entero:

- `add` *`rd, rs, rt`*

  Almacena la suma del contenido de los registros *`rs`* y *`rt`* en el registro *`rd`*.

- `addi` *`rd, rs, entero`*

  Almacena la suma del contenido del registro *`rs`* y *`entero`* en el registro *`rd`*.

- `sub` *`rd, rs, rt`*

  Almacena la diferencia entre el contenido de los registros *`rs`* y *`rt`* en el registro *`rd`*.

- `subi` *`rd, rs, entero`*

  Almacena la diferencia del contenido del registro *`rs`* y *`entero`* en el registro *`rd`*.

- `mult` *`rd, rs, rt`*

  Almacena el producto del contenido de los registros *`rs`* y *`rt`* en el registro *`rd`*.

- `multi` *`rd, rs, entero`*

  Almacena el producto del contenido del registro *`rs`* y *`entero`* en el registro *`rd`*.

- `div` *`rd, rs, rt`*

  Almacena el cociente del contenido de los registros *`rs`* y *`rt`* en el registro *`rd`*.

- `divi` *`rd`*`,` *`rs`*`,` *`entero`*

  Almacena el cociente del contenido del registro *`rs`* y *`entero`* en el registro *`rd`*.

- `mod` *`rd`*`,` *`rs`*`,` *`rt`*

  Almacena el módulo del contenido de los registros *`rs`* y *`rt`* en el registro *`rd`*.

- `modi` *`rd`*`,` *`rs`*`,` *`entero`*

  Almacena el módulo del contenido del registro *`rs`* y *`entero`* en el registro *`rd`*.

- `not` *`rd`*`,` *`rs`*

  Almacena un 1 en *`rd`* si *`rs`* contiene un 0. En caso contrario se almacena un 0.

Las siguientes instrucciones aritméticas exigen que sus operandos sean de tipo real:

- `fadd` *`rd`*`,` *`rs`*`,` *`rt`*

  Almacena la suma del contenido de los registros *`rs`* y *`rt`* en el registro *`rd`*.

- `faddi` *`rd`*`,` *`rs`*`,` *`real`*

  Almacena la suma del contenido del registro *`rs`* y *`real`* en el registro *`rd`*.

- `fsub` *`rd`*`,` *`rs`*`,` *`rt`*

  Almacena la diferencia entre el contenido de los registros *`rs`* y *`rt`* en el registro *`rd`*.

- `fsubi` *`rd`*`,` *`rs`*`,` *`real`*

  Almacena la diferencia del contenido del registro *`rs`* y *`real`* en el registro *`rd`*.

- `fmult` *`rd`*`,` *`rs`*`,` *`rt`*

  Almacena el producto del contenido de los registros *`rs`* y *`rt`* en el registro *`rd`*.

- `fmulti` *`rd`*`,` *`rs`*`,` *`real`*

  Almacena el producto del contenido del registro *`rs`* y *`real`* en el registro *`rd`*.

- `fdiv` *`rd`*`,` *`rs`*`,` *`rt`*

  Almacena el cociente del contenido de los registros *`rs`* y *`rt`* en el registro *`rd`*.

- `fdivi` *`rd`*`,` *`rs`*`,` *`real`*

  Almacena el cociente del contenido del registro *`rs`* y *`real`* en el registro *`rd`*.

5.2. Instrucciones de carga y almacenamiento

Las siguientes instrucciones, destinadas a la carga y almacenamiento de valores enteros, exigen que el primer operando sea un registro entero:

- `la` *`rd`*`,` *`etiqueta`*

  Almacena la línea en la que se define *`etiqueta`* en el registro *`rd`*.

- `sw` *`rd`*`,` *`direccion`*

  Almacena el contenido del registro *`rd`* en la posición de memoria que indica *`direccion`*.

- `save` *`rd`*`,` *`direccion`*

  Almacena el contenido del registro *`rd`* en la posición de memoria que indica *`direccion`*. En caso de que *`rd`* no se encuentre inicializado, no se produce ninguna excepción. Esta instrucción impone la restricción de que el registro entero utilizado para calcular *`direccion`* debe ser `$sp`.

  Al igual que sucede con la instrucción `fsave`, esta instrucción está diseñada para poder salvar el contenido de un registro que va a ser modificado por una subrutina sin necesidad de inicializar previamente dicho registro.

- `lw` *`rd`*`,` *`direccion`*

  Almacena el valor que contiene la posición de memoria especificada por *`direccion`* en el registro *`rd`*.

- `rest` *`rd`*`,` *`direccion`*

  Almacena el valor que contiene la posición de memoria especificada por *`direccion`* en el registro *`rd`*. En caso de que la posición de memoria no contenga ningún valor, no se produce ninguna excepción. Esta instrucción impone la restricción de que el registro entero utilizado para calcular *`direccion`* debe ser `$sp`.

  Esta instrucción está diseñada para poder restaurar el contenido de un registro que ha sido apilado con la instrucción `save`.

Las siguientes instrucciones, destinadas a la carga y almacenamiento de valores reales, exigen que el primer operando sea un registro real:

- `fsw` *`rd`*`,` *`direccion`*

  Almacena el contenido del registro *`rd`* en la posición de memoria que indica *`direccion`*.

- `fsave` *`rd`*`,` *`direccion`*

  Almacena el contenido del registro *`rd`* en la posición de memoria que indica *`direccion`*. En caso de que *`rd`* no se encuentre inicializado, no se produce

ninguna excepción. Esta instrucción impone la restricción de que el registro entero utilizado para calcular `direccion` debe ser `$sp`.

- `flw` *`rd`*, *`direccion`*

  Almacena el valor que contiene la posición de memoria especificada por *`direccion`* en el registro *`rd`*.

- `frest` *`rd`*, *`direccion`*

  Almacena el valor que contiene la posición de memoria especificada por *`direccion`* en el registro *`rd`*. En caso de que la posición de memoria no contenga ningún valor, no se produce ninguna excepción. Esta instrucción impone la restricción de que el registro entero utilizado para calcular *`direccion`* debe ser `$sp`.

  Esta instrucción está diseñada para poder restaurar el contenido de un registro que ha sido apilado con la instrucción `fsave`.

5.3. Instrucciones de salto condicional

Las siguientes instrucciones exigen que los registros sean de tipo entero:

- `beq` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define *`etiqueta`* si el contenido de los registros *`rd`* y *`rs`* es el mismo.

- `bne` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define *`etiqueta`* si el contenido de los registros *`rd`* y *`rs`* es distinto.

- `bge` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define *`etiqueta`* si el contenido del registro *`rd`* es mayor o igual que el contenido del registro `rs`.

- `bgt` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define *`etiqueta`* si el contenido del registro *`rd`* es estrictamente mayor que el contenido del registro *`rs`*.

- `ble` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define *`etiqueta`* si el contenido del registro *`rd`* es menor o igual que el contenido del registro *`rs`*.

- `blt` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define *`etiqueta`* si el contenido del registro *`rd`* es estrictamente menor que el contenido del registro *`rs`*.

Las siguientes instrucciones exigen que los registros sean de tipo real:

- `fbeq` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define `etiqueta` si el contenido de los registros *`rd`* y *`rs`* es el mismo.

- `fbne` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define `etiqueta` si el contenido de los registros *`rd`* y *`rs`* es distinto.

- `fbge` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define `etiqueta` si el contenido del registro *`rd`* es mayor o igual que el contenido del registro *`rs`*.

- `fbgt` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define *`etiqueta`* si el contenido del registro *`rd`* es estrictamente mayor que el contenido del registro *`rs`*.

- `fble` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define `etiqueta` si el contenido del registro *`rd`* es menor o igual que el contenido del registro *`rs`*.

- `fblt` *`rd`*, *`rs`*, *`etiqueta`*

  Salta a la línea donde se define *`etiqueta`* si el contenido del registro *`rd`* es estrictamente menor que el contenido del registro *`rs`*.

5.4. Instrucciones de salto incondicional

- `j` *`etiqueta`*

  Salta a la línea donde se define `etiqueta`.

- `jal` *`etiqueta`*

  Almacena la dirección de la siguiente instrucción en el registro `$ra` y salta a la línea donde se define `etiqueta`.

- `jr` *`rd`*

  Salta a la instrucción cuya dirección está almacenada en el registro entero *`rd`*.

- `jalr` *`rd`*

  Almacena la dirección de la siguiente instrucción en el registro `$ra` y salta a la instrucción cuya dirección está almacenada en el registro entero `rd`.

5.5. Instrucciones de conversión de tipos

- `toint` *`rd, rs`*

  Convierte a entero el contenido del registro real *`rs`* y lo almacena en el registro entero *`rd`*.

- `tofloat` *`rd, rs`*

  Convierte a real el contenido del registro entero *`rs`* y lo almacena en el registro real *`rd`*.

5.6. Llamadas al sistema

- `syscall`

  Esta instrucción sirve para realizar una llamada al sistema en la máquina **ROSSI**, bien para realizar una operación de entrada/salida o bien para finalizar la ejecución del programa de modo normal. Para solicitar un servicio se debe cargar el código de la llamada en el registro `$sc` y los argumentos, en los registros correspondientes (en caso de que se necesiten).

  La siguiente tabla muestra cuáles son los servicios ofrecidos, indicando el código de cada uno de ellos, sus argumentos y el resultado proporcionado.

Servicio	Código	Argumentos	Resultado
`imprimir entero`	0	entero en `$a0`	—
`imprimir real`	1	real en `$fa`	—
`imprimir cadena`	2	dirección en `$a0`	—
`leer entero`	3	—	entero en `$a0`
`leer real`	4	—	real en `$fa`
`leer cadena`	5	dirección en `$a0` longitud en `$a1`	cadena en memoria
`exit`	6	—	—

Tabla 4. Servicios proporcionados con la instrucción `syscall`.

Los servicios 0 y 1 imprimen en el terminal de entrada/salida el valor numérico correspondiente. El formato de salida de este valor dependerá del emulador de la máquina que se utilice.

El servicio 2 imprime los caracteres cuyos códigos ASCII se encuentran almacenados en memoria a partir de la dirección especificada, hasta encontrar el carácter nulo.

Los servicios 3 y 4 leen el valor numérico introducido por teclado, hasta encontrar el primer retorno de carro, y lo almacenan en el registro correspondiente. El formato del valor introducido vendrá determinado por el emulador de la máquina que se esté utilizando.

El servicio 5 lee la cadena introducida por teclado hasta encontrar el primer retorno de carro, que no forma parte de la cadena. A partir de la dirección especificada en la llamada, se van almacenando los códigos ASCII correspondientes a cada uno de los caracteres que forman la cadena leída, hasta almacenar un número de caracteres igual al mínimo entre la longitud especificada y la longitud de la cadena leída. Tras almacenar la cadena en memoria se marca su final añadiéndole el carácter nulo.

El servicio 6 es el utilizado para finalizar la ejecución de cualquier programa de modo normal. Si se alcanza el final del fichero sin haber ejecutado este servicio, la unidad de control producirá una excepción.

6. Excepciones

Existen determinadas situaciones en las que la ejecución de una instrucción puede provocar una excepción en la unidad de control, dejando la máquina detenida en un estado de error. Cuando se produce una excepción, al no haber finalizado la ejecución de la instrucción en curso, el registro `$pc` no llega a incrementarse, por lo que, en este caso, su contenido es la dirección de la instrucción que ha provocado la excepción.

Las diferentes excepciones que pueden suceder en la máquina virtual **ROSSI** se clasifican en las siguientes categorías:

- **Relacionadas con llamadas al sistema:** este tipo de excepciones sucede cuando se realiza una llamada al sistema sin especificar el código de la llamada, cuando se proporciona un código de llamada no válido, cuando se intenta leer un entero o un real y el valor introducido por teclado no tiene el formato esperado o cuando se pretende escribir un carácter con código ASCII no comprendido entre 0 y 255.

- **Relacionadas con registros:** se producen cuando se intenta acceder a un registro para lectura y éste no contiene ningún valor. Inicialmente todos los registros están vacíos, excepto los registros `$zero` y `$fzero`, que contienen los valores 0 y 0.0, respectivamente.

- **Relacionadas con memoria:** estas excepciones suceden cuando se intenta leer una posición de memoria que no contiene ningún valor, cuando se intenta leer un entero de una posición de memoria que contiene un real, o viceversa.

- **Relacionadas con operaciones de división o módulo:** este tipo de excepciones aparece cuando se intenta realizar una división o un módulo con divisor cero.

- **Relacionadas con saltos incondicionales:** estas excepciones las lanza la unidad de control cuando el destino de una instrucción de salto `jr` no es la instrucción posterior a ninguna llamada a subrutina (instrucción `jal o jalr`) ni está etiquetado. De este modo se pretende tener un mayor control de errores de programación relacionados con saltos.

- **Relacionadas con el PC (PC fuera de rango):** este tipo de excepciones sucede cuando se alcanza el final del fichero sin haber ejecutado el servicio `exit` proporcionado por la máquina.

7. Programas de ejemplo

En este apartado se presentan diferentes programas de ejemplo para la máquina virtual **ROSSI**.

7.1. Hola mundo

El siguiente programa `hola.rossi` imprime la cadena `Hola mundo!` por pantalla, seguida de un salto de línea.

```
.data
.asciiz "Hola mundo!\n"

.text
addi   $sc, $zero, 2       # Código de imprimir cadena
addi   $a0, $zero, 0       # Dirección de la cadena
syscall
addi   $sc, $zero, 6       # exit
syscall
```

7.2. Conversor de temperaturas

El programa `temp.rossi` que se muestra a continuación lee la temperatura en grados Celsius introducida por el usuario y la convierte en grados Fahrenheit, imprimiendo el resultado por pantalla.

```
.data
.asciiz "Introduce la temperatura (grados Celsius): "
.asciiz 50 "La temperatura en grados Fahrenheit es "
.asciiz 100 "\n"

.text
addi   $sc, $zero, 2
addi   $a0, $zero, 0
syscall                    # Solicita la temperatura
addi   $sc, $zero, 4
syscall                    # Lee un real en $fa
fmulti $f0, $fa, 9.0       # Para convertir, multiplicar por 9
fdivi  $f0, $f0, 5.0       # dividir por 5 y
faddi  $f0, $f0, 32.0      # sumar 32
```

```
        addi   $sc, $zero, 2
        addi   $a0, $zero, 50
        syscall
        addi   $sc, $zero, 1
        fadd   $fa, $f0, $fzero
        syscall                   # Imprimime el resultado
        addi   $sc, $zero, 2
        addi   $a0, $zero, 100
        syscall                   # Imprime salto de línea
        addi   $sc, $zero, 6
        syscall                   # Fin del programa
```

7.3. Cálculo de cuadrados

El programa `cuadrado.rossi` calcula los cuadrados de los *N* primeros números enteros y los imprime por pantalla.

```
        .data
        .asciiz "El cuadrado de "
        .asciiz 20 " es "
        .asciiz 25 "\n"
        .asciiz 30 "N: "

        .text
        addi   $sc, $zero, 2
        addi   $a0, $zero, 30
        syscall
        addi   $sc, $zero, 3
        syscall
        add    $r0, $a0, $zero     # Copia el límite en $r0
        addi   $r1, $zero, 1       # $r1 = Índice
cond:   bgt    $r1, $r0, fin
        mult   $r2, $r1, $r1       # Cuadrado en $r2
        addi   $sc, $zero, 2
        addi   $a0, $zero, 0
        syscall
        addi   $sc, $zero, 0
        add    $a0, $r1, $zero
        syscall
        addi   $sc, $zero, 2
        addi   $a0, $zero, 20
        syscall
        addi   $sc, $zero, 0
```

```
        add    $a0, $r2, $zero
        syscall
        addi   $sc, $zero, 2
        addi   $a0, $zero, 25
        syscall
        addi   $r1, $r1, 1  # Incrementa el índice
        j      cond
fin:    addi   $sc, $zero, 6
        syscall
```

7.4. Función de Fibonacci

El último programa de ejemplo, `fibo.rossi`, calcula el valor de Fibonacci(*N*) de forma recursiva para cualquier *N* introducido por el usuario.

```
        .data
        .asciiz    "N: "
        .asciiz 5  "\n"
        .asciiz 10 "fibo("
        .asciiz 20 ") = "

        .text
        addi   $sc, $zero, 2
        addi   $a0, $zero, 0
        syscall
        addi   $sc, $zero, 3
        syscall
        add    $r3, $a0, $zero     # N en $r3
        addi   $sp, $zero, 100     # Inicialización del stack pointer
        addi   $sp, $sp, 1         # Creamos espacio para el resultado
        sw     $r3, 0($sp)         # Apilamos N
        addi   $sp, $sp, 1         # Incrementamos el $sp
        jal    fibo
        addi   $sc, $zero, 2
        addi   $a0, $zero, 10
        syscall
        addi   $sc, $zero, 0
        add    $a0, $r3, $zero
        syscall
        addi   $sc, $zero, 2
        addi   $a0, $zero, 20
        syscall
        lw     $a0, 0($sp)         # Resultado, de la cima de la pila a $a0
```

```
        addi   $sc, $zero, 0
        syscall
        addi   $sc, $zero, 2
        addi   $a0, $zero, 5
        syscall
        addi   $sc, $zero, 6
        syscall                   # exit
        # La función deberá apilar el valor del registro $r0 para no perder su
        # valor en sucesivas llamadas recursivas. No es necesario apilar $r1 ni
        # $r2 puesto que, en este caso, se usan como temporales.
fibo:   save   $fp, 0($sp)        # Apilamos el $fp
        addi   $fp, $sp, 0        # El $fp actual apunta al $fp anterior
        addi   $sp, $sp, 1
        sw     $ra, 0($sp)        # Apilamos la dirección de retorno
        addi   $sp, $sp, 1
        save   $r0, 0($sp)        # Salvamos el contenido del registro $r0
        addi   $sp, $sp, 1
        lw     $r0, -1($fp)       # N en $r0
        addi   $r1, $zero, 1
        ble    $r0, $r1, uno      # Si N <= 1 saltamos a "uno"
        # Llamamos a fibo(n-1)
        addi   $sp, $sp, 1        # Creamos espacio para el resultado
        subi   $r2, $r0, 1        # $r2 = N-1
        sw     $r2, 0($sp)
        addi   $sp, $sp, 1
        jal    fibo
        # Llamamos a fibo(n-2)
        subi   $r2, $r0, 2        # $r2 = N-2
        lw     $r0, 0($sp)        # $r0 = fibo(n-1)
        addi   $sp, $sp, 1
        sw     $r2, 0($sp)
        addi   $sp, $sp, 1
        jal    fibo
        lw     $r2, 0($sp)        # $r2 = fibo(n-2)
        add    $r0, $r0, $r2      # $r0 = fibo(n-1) + fibo(n-2)
        sw     $r0, -2($fp)
        j      fin
uno:    sw     $r1, -2($fp)       # Devolvemos un 1
fin:    subi   $sp, $sp, 5        # Restauramos el $sp (apunta al resultado)
        lw     $ra, 1($fp)        # Restauramos la dirección de retorno
        rest   $r0, 4($sp)        # Restauramos el registro $r0
        rest   $fp, 2($sp)        # Restauramos el $fp
        jr     $ra
```

8. TheDoc: un emulador para la máquina virtual ROSSI

El emulador **TheDoc**, implementado en su totalidad en el lenguaje de programación Python (versión 3.3.2), permite la carga, ejecución y seguimiento de programas **ROSSI**, emulando el comportamiento de la máquina virtual.

El emulador, directamente ejecutable desde el intérprete de órdenes de Linux mediante la orden `thedoc`, ofrece tres modos de funcionamiento distintos: modo batch, modo texto y modo gráfico, correspondientes a las opciones `-b`, `-t` y `-g`, respectivamente. El funcionamiento por defecto es en modo gráfico, y el modo texto es activado de forma automática si se produce cualquier error al cargar las librerías gráficas de Python (Pmw 1.2 o Tkinter) o si existe cualquier problema con el servidor gráfico de la máquina sobre la cual se está ejecutando. Además de las opciones anteriormente mencionadas, el emulador acepta en su invocación un argumento de carácter opcional que, en caso de que se especifique, determina el programa que se debe cargar al inicio.

Cabe mencionar en este punto que el formato de los valores numéricos aceptados por el emulador en las operaciones de lectura vendrá determinado por la versión de Python que se esté utilizando.

Para poder comenzar la ejecución de un programa, independientemente de cuál sea el modo de funcionamiento seleccionado, es condición necesaria que esté libre de errores de análisis. En caso contrario, el emulador indicará al usuario la línea o líneas incorrectas, informando del error sucedido con la mayor precisión posible.

Durante la ejecución de un programa **ROSSI** puede suceder que la máquina quede detenida en un estado de error, como consecuencia de una excepción. En ese caso, el emulador informará al usuario de la excepción sucedida, indicando la instrucción que ha provocado la excepción.

Tras finalizar la ejecución de un programa de modo normal, el emulador presenta una serie de resultados, por la salida de error estándar, que permiten obtener una aproximación de la eficiencia del programa. Estos resultados son:

- Cantidad de registros enteros de propósito general (REPG) usados.
- Cantidad de registros reales de propósito general (RRPG) usados.
- Número de pasos necesarios para ejecutar el programa.

En los siguientes apartados se describe cada uno de los modos de funcionamiento ofrecidos por el emulador.

8.1. Ejecución en modo batch

La invocación del emulador en modo batch tiene como resultado la carga y ejecución del programa que se ha especificado como argumento, sin ninguna interacción por parte del usuario (salvo la entrada por teclado, si el programa lo requiere).

A continuación se muestra la ejecución del programa `fibo.rossi` presentado con anterioridad:

```
$ ./thedoc -b ../samples/fibo.rossi
N: 17
fibo(17) = 2584

[ Ejecución finalizada ]
REPG usados:   4
RRPG usados:   0
Pasos: 118868
```

En caso de no haber especificado el programa que se debe ejecutar, el emulador lee el programa de la entrada estándar hasta encontrar el carácter fin de fichero (`^D`):

```
$ ./thedoc -b
.text
add     $r0, $zero, et
.asciiz "adiós"
^D
Se han encontrado los siguientes errores:
Error línea 1: El tercer argumento de la instrucción 'add' debe ser un registro entero.
Error línea 2: la directiva '.asciiz' sólo puede aparecer en la zona de definición de datos.
```

Este modo de funcionamiento es el indicado cuando únicamente se desea llevar a cabo la ejecución del programa, sin necesidad de conocer el contenido de los registros ni la memoria.

8.2. Ejecución en modo texto

La ejecución del emulador en modo texto inicia una sesión interactiva que permite realizar un seguimiento del programa: visualizar el contenido de los registros y la memoria a medida que avanza la ejecución, fijar puntos de ruptura, permitir la ejecución paso a paso, etcétera.

Al iniciar el emulador, se muestra un mensaje de bienvenida como el siguiente:

```
$ ./thedoc -t
Bienvenido a TheDoc: un emulador para la máquina virtual ROSSI.
Escribe 'help' para mostrar las órdenes disponibles.
>>
```

tras el cual, se procesan cada una de las órdenes introducidas por el usuario hasta que finalice la sesión. Las órdenes de las que dispone **TheDoc** son las siguientes:

- **`load file`:** carga el programa listado en el fichero `file`.
- **`reload`:** recarga el último programa cargado, inicializa el contenido de los registros y la memoria, y elimina los breakpoints definidos.
- **`step`:** avanza un paso en la ejecución del programa.
- **`execute`:** ejecuta el programa hasta finalizar la ejecución de modo normal, alcanzar un punto de ruptura (breakpoint) o producirse una excepción.
- **`breakpoint [n]`:** inserta o elimina un punto de ruptura en la línea `n`. Si no se ha especificado `n`, muestra todos los breakpoints que se han definido sobre el programa actual.
- **`program [n [m]]`:** muestra el programa cargado actualmente desde la línea `n` hasta la *m* indicando, mediante un asterisco (`*`), la instrucción que se va a ejecutar en el siguiente paso. Si no se especifica *m*, muestra el programa a partir de la línea `n`. Si tampoco se especifica `n`, muestra el programa completo.
- **`memory [n [m]]`:** muestra el contenido de la memoria desde la posición `n` hasta la *m*. Si no se especifica *m*, muestra el contenido de la memoria a partir de la posición `n`. Si tampoco se especifica *n*, muestra el contenido de toda la memoria.
- **`integers [n [m]]`:** muestra el contenido de los registros enteros de propósito general desde el registro `$rn` hasta el `$rm`. Si no se especifica *m*, muestra el contenido de los registros enteros de propósito general a partir del registro `$rn`. Si tampoco se especifica `n`, muestra el contenido de todos los registros enteros de propósito general.
- **`reals [n [m]]`:** muestra el contenido de los registros reales de propósito general desde el registro `$fn` hasta el `$fm`. Si no se especifica *m*, muestra el contenido de los registros reales de propósito general a partir del registro `$rn`. Si tampoco se especifica `n`, muestra el contenido de todos los registros reales de propósito general.
- **`spints`:** muestra el contenido de los registros enteros de propósito específico.
- **`spreals`:** muestra el contenido de los registros reales de propósito específico.
- **`verbose`:** activa o desactiva la verbosidad del emulador. Si se encuentra activada, se muestra la instrucción que se ejecuta en cada paso y las modificaciones sufridas en los bancos de registros o memoria.
- **`reset`:** inicializa el contenido de los registros y la memoria, manteniendo cargado el programa actual.
- **`help`:** muestra las órdenes disponibles por pantalla.

- **`exit`, `^D`, `^C`:** finaliza la sesión actual.

Cabe mencionar que, para ejecutar una orden, **TheDoc** no requiere que ésta se escriba de forma completa. Basta con introducir un prefijo que no coincida con el de ninguna otra orden disponible.

A continuación, se muestra una breve sesión de ejemplo:

```
>> load ../samples/hola.rossi
>> br 10
>> pr
*         0| # Escribe hola mundo en el terminal de entrada/salida
          1|
          2|    .data
          3|
          4|    .asciiz "Hola mundo!\n"
          5|
          6|    .text
          7|
          8|    addi    $sc, $zero, 2   # Código de imprimir cadena
          9|    addi    $a0, $zero, 0   # Dirección de la cadena
   BRK   10|    syscall
         11|    addi    $sc, $zero, 6   # exit
         12|    syscall
>> mem
>> exe
>> pr
          0| # Escribe hola mundo en el terminal de entrada/salida
          1|
          2|    .data
          3|
          4|    .asciiz "Hola mundo!\n"
          5|
          6|    .text
          7|
          8|    addi    $sc, $zero, 2   # Código de imprimir cadena
          9|    addi    $a0, $zero, 0   # Dirección de la cadena
*  BRK   10|    syscall
         11|    addi    $sc, $zero, 6   # exit
         12|    syscall
>> mem
    0 : 72
    1 : 111
    2 : 108
```

```
     3 : 97
     4 : 32
     5 : 109
     6 : 117
     7 : 110
     8 : 100
     9 : 111
    10 : 33
    11 : 10
    12 : 0
>> exe
Hola mundo!

[ Ejecución finalizada ]
REPG usados:    0
RRPG usados:    0
Pasos: 8
>> exit
Hasta otra!
```

8.3. Ejecución en modo gráfico

El modo gráfico es, como se ha comentado anteriormente, el modo de ejecución por defecto de **TheDoc**. La invocación del emulador con este modo de funcionamiento genera un entorno gráfico funcional que permite realizar un control y seguimiento de los programas **ROSSI** de un modo visual e intuitivo.

La figura 1 muestra el aspecto gráfico de **TheDoc** e indica cada una de las zonas significativas de la aplicación (1-9).

A continuación, se describe detalladamente la funcionalidad de cada una de estas zonas:

1. **Barra de menús:** la barra de menús permite navegar por los diferentes menús de la aplicación y seleccionar cualquiera de sus opciones. **TheDoc** presenta los dos menús siguientes: `File` y `About`.

   El menú `File` contiene las siguientes opciones:

   - `Load program`: efectúa la carga del programa, seleccionado mediante un diálogo de selección de ficheros con extensión `.rossi`, inicializando además el contenido de las zonas de registros y memoria, el terminal de entrada/salida, y eliminando los breakpoints definidos.

   - `Reload program`: recarga el programa cargado actualmente, inicializa las zonas de registros y memoria, el terminal de entrada/salida, y elimina los breakpoints definidos. Esta opción permanece desactivada hasta que se realiza la carga de un programa.

El menú `About` contiene una única opción, `About TheDoc`, que permite obtener información general sobre el programa.

La barra de menús permanece desactivada mientras el emulador está ejecutando el programa de modo contínuo.

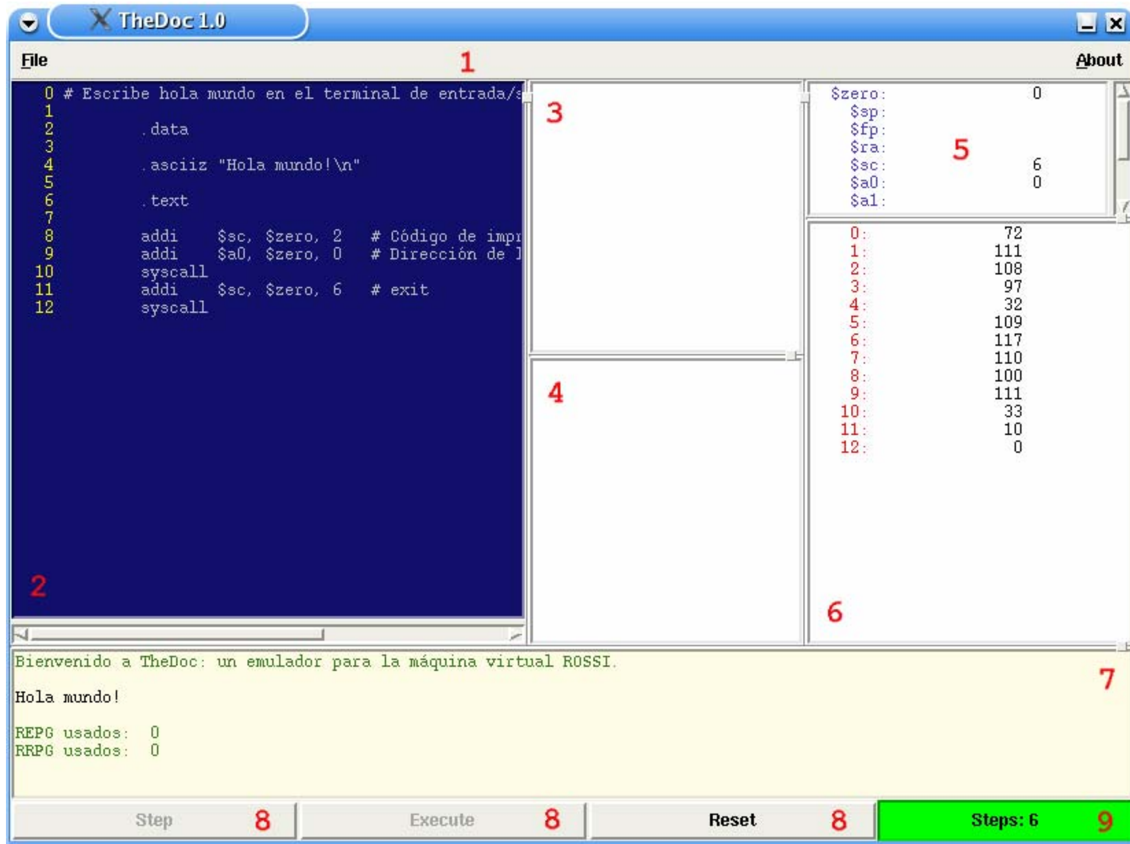

Figura 1. Interfaz gráfica del emulador **TheDoc**.

2. **Zona del programa:** en esta zona se muestra el programa cargado actualmente. Cada instrucción va precedida del número de línea en la que aparece, para facilitar la localización de instrucciones.

   La instrucción que se va a ejecutar en cada paso se distingue de las demás por tener fondo amarillo. Una línea con un punto de ruptura definido se caracteriza por tener el número de línea sobre fondo naranja.

   La introducción o eliminación de un punto de ruptura sobre una línea consiste en hacer un simple click con el ratón sobre el número de línea.

3. **Zona de los registros enteros de propósito general:** en esta zona se muestra el contenido de los registros enteros de propósito general utilizados a medida que avanza la ejecución del programa. Siempre se muestra el último registro modificado.

4. **Zona de los registros reales de propósito general:** en esta zona se muestra el contenido de los registros reales de propósito general utilizados a medida que avanza la ejecución del programa. Siempre se muestra el último registro modificado.

5. **Zona de los registros de propósito específico:** en esta zona se muestra el contenido de los registros de propósito específico (tanto enteros como reales) a medida que avanza la ejecución del programa.

6. **Zona de memoria:** en esta zona se muestra el contenido de la memoria de datos utilizada a medida que avanza la ejecución del programa. Siempre se muestra el contenido de la última posición de memoria actualizada.

7. **Terminal de entrada/salida:** el terminal de entrada/salida tiene tres funciones claramente diferenciables:

   - Mostrar el mensaje de bienvenida y, tras la ejecución de un programa que no ha provocado ninguna excepción, los valores de REPG y RRPG usados. Este uso del terminal se diferencia de los demás por usar una fuente de color verde.

   - Mostrar los errores de análisis encontrados al cargar el programa. En este caso, la fuente usada para escribir los mensajes es de color rojo.

   - Emular el terminal de entrada/salida de la máquina **ROSSI**. La fuente utilizada cuando está efectuando esta función es de color negro.

8. **Botones de acción:** desde esta zona se realizan las acciones principales del emulador. Los botones de acción aparecen inicialmente desactivados, y la activación/desactivación de cada uno de ellos se realiza en función de la posibilidad de aplicar la acción correspondiente en el momento actual. Existen tres botones que se corresponden con las acciones básicas:

   - `Step`: avanza un paso en la ejecución del programa.

   - `Execute`: ejecuta el programa de modo continuo hasta finalizar el programa de modo normal, encontrar un breakpoint o producirse una excepción. Mientras se está ejecutando el programa de forma continua el botón `Execute` cambia de funcionalidad y pasa a convertirse en `Stop execution`, que tiene como acción asociada detener la ejecución del programa en curso.

   - `Reset`: inicializa la máquina, las zonas de registros y memoria y el terminal de entrada/salida.

9. **Contador de pasos:** muestra el número de instrucciones ejecutadas durante la ejecución del programa. El número de pasos intermedios se muestra sobre fondo blanco. El número de pasos totales empleados en ejecutar el programa se indica sobre fondo verde.