Práctica 12. El simulador CREATOR

Objetivos

- Conocer CREATOR, el simulador de RISC-V.
- Conocer las instrucciones aritméticas y lógicas en lenguaje ensamblador de RISC-V.

1. Introducción

Las prácticas de ensamblador de la asignatura de Estructura de Computadores consisten en el estudio detallado de la arquitectura RISC-V para comprobar los conocimientos impartidos en las clases teóricas.

2. Introducción al simulador RARS

La arquitectura RISC-V fue desarrollada originalmente en la Universidad de California en Berkeley (2010) y se basa en un diseño de tipo RISC cuya arquitectura es de tipo Open Source. Por tanto puede ser utilizada por cualquier persona o compañía sin pagar ningún canon. Recientemente se ha desarrollado un chip con arquitectura RISC-V denominado EPAC (European Processor ACcelerator) [1]. Desde el Barcelona Supercomputing Center (BSC) se está fomentando el uso de procesadores basados en la arquitectura RISC-V para conseguir evitar la dependencia existente con los procesadores americanos o de otros países[2].

En el laboratorio, se utilizará un simulador denominado CREATOR (didaCtic and geneRic assEmbly prograMming simulaTOR) desarrollado por profesores de la Universidad Carlos III de Madrid, libre y multiplataforma. Su funcionamiento es sencillo, e incluye un depurador que, entre otras muchas funciones,

[1] https://architecnologia.es/risc-v-el-nuevo-chip-europeo-rhea-y-primer-fruto-de-epi

[2] https://riscv.org/

permite ejecutar los programas paso a paso y muestra el estado de los registros.

El programa CREATOR está disponible en `https://creatorsim.github.io/creator/`.

3. Funcionamiento del simulador

En la dirección indicada (Figura 1), veremos la opción de usar el simulador de una arquitectura RISC-V o de una arquitectura MIPS; escogemos la arquitectura RISC-V.

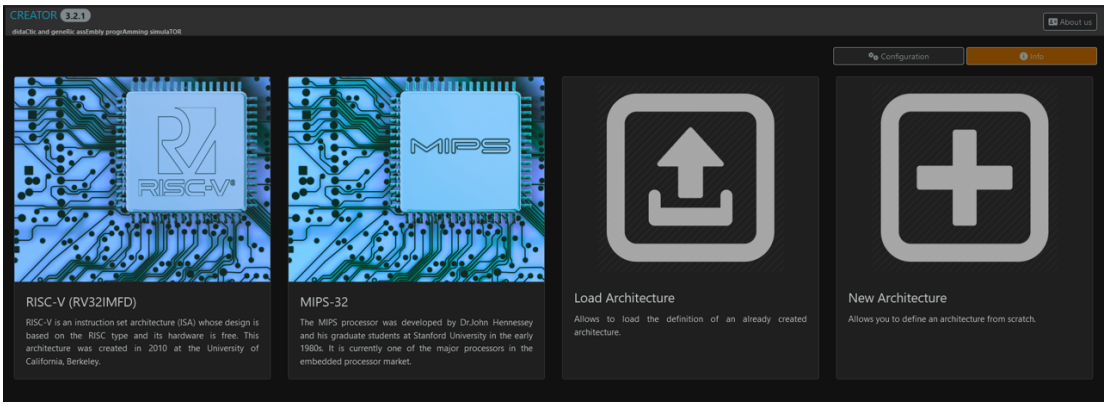

Figura 1: Página principal del simulador

Una vez en el simulador, vemos que se distinguen cuatro partes (ver Figura 2):

3.1. Panel de Menús/Botones

Muestra las operaciones que se pueden realizar. Según los botones es posible:

- Elegir la arquitectura de otro procesador (botón *Architecture*, Fig. 3).
- Alternar entre la edición y la ejecución de programas (botón *#Assembly* y botón *#Simulator*, Fig. 3).
- Resetear el simulador (botón *Reset*, Fig. 3).
- Ejecutar paso a paso (botón *Instr.*, Fig. 3).
- Ejecutar el programa entero (botón *Run*, Fig. 3).

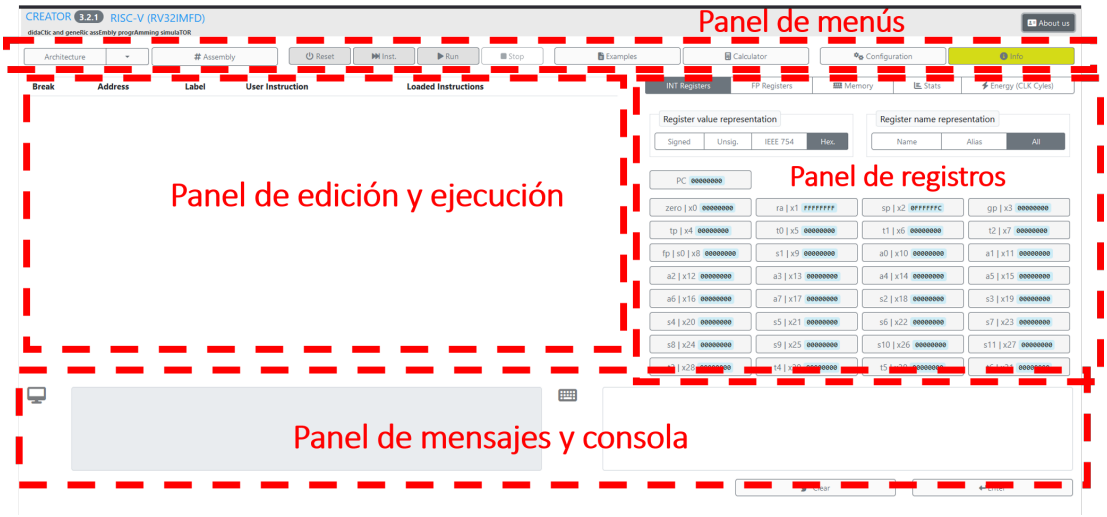

Figura 2: Ventana principal del simulador

Figura 3: Botones disponibles para editar y ejecutar programas

- Parar el programa que se está ejecutando (botón *Stop*, Fig. 3).
- Ver ejemplos disponibles (botón *Examples*, Fig. 4).
- Ver cómo se representa un número real en IEEE754 (botón *Calculator*, Fig. 4).
- Configurar el simulador (botón *Configuration*, Fig. 4).
- Obtener información sobre las instrucciones RISC-V o sobre el simulador con recursos de aprendizaje (botón *Info*, Fig. 4).

Figura 4: Botones disponibles para ver ejemplos, configuración y más información

3.2. Panel de Registros

Muestra los registros del procesador 5, tanto los de propósito general (`$zero`, ... `$t0`, `$t1`, ...), el registro `$ra`, el registro `$sp`, como el registro contador de programa `PC` que indica qué instrucción se va a ejecutar. Se indica su nombre pero también a qué registro corresponde (`x0, x1, x2, ... x31`).

Se puede visualizar el contenido de los registros como valores con signo, sin signo, en el estándar IEEE754 (solo apropiado para reales) y en hexadecimal. También se puede escoger entre mostrar el nombre de los registros, su alias o ambas cosas.

Figura 5: Registros de propósito general y contador de programa (PC)

Si presionamos el botón *FP Registers* aparecen los registros de coma flotante (Fig. 6), con las mismas opciones que para los registros de propósito general.

3.3. Panel de Mensajes y Consola

Muestra los mensajes de información del simulador y la consola o interfaz del usuario. Por tanto, en esta zona de la pantalla se muestran los mensajes de los programas y se leen datos del teclado.

3.4. Panel de Edición y Ejecución

Si damos al botón *#Assembly* pasamos al panel de Edición donde podemos escribir programas en lenguaje ensamblador (ver Fig. 7). Con el botón *Compile/Linked* se puede compilar y nos dirá si hay errores.

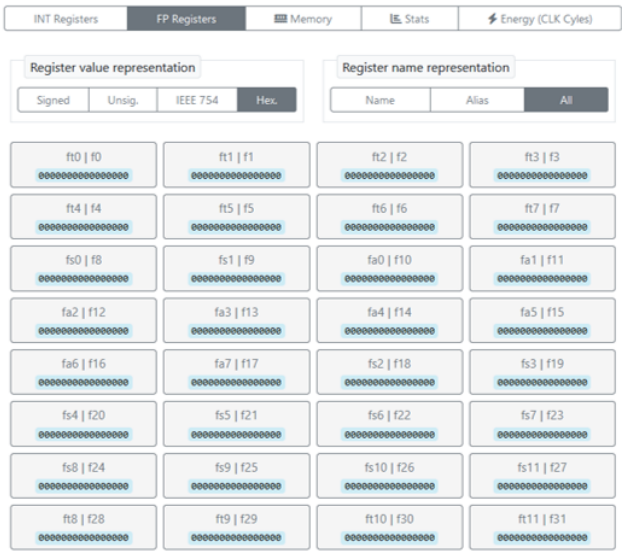

Figura 6: Registros de coma flotante

En el botón *File* se puede crear un nuevo archivo, cargar un archivo existente o guardar el programa. Además se pueden cargar los ejemplos disponibles y obtener la URL de un ejercicio, muy útil para compartir código.

Con el botón *Library* es posible crear código y cargarlo para que se pueda utilizar en el programa que se está escribiendo, de forma análoga al uso de `include` en el lenguaje C.

Una vez compilado el programa con el botón *Compile/Linked*, si no ha dado error, es posible seleccionar el botón *Simulator* para observar cómo se ha cargado el programa. Como se muestra en la figura 8, hay varias columnas:

- *Break*: si se hace clic se añade/elimina un punto de ruptura.
- *Address*: la dirección donde se ha cargado la instrucción. Como las instrucciones ocupan 32 bits (4 bytes) se almacenan en posiciones múltiplo de 4 (en hexadecimal acaban en 0, 4, 8 o C).
- *Label*: si esa instrucción tiene una etiqueta.
- *User instruction*: la instrucción escrita en el programa.
- *Loaded instruction*: la instrucción que se carga en el procesador. Si se ha empleado una instrucción que el procesador reconoce, coincidirá con la indicada por el usuario. Si se ha empleado una pseudoinstrucción, se traduce en la instrucción que entiende el procesador.

Figura 7: Edición de programas

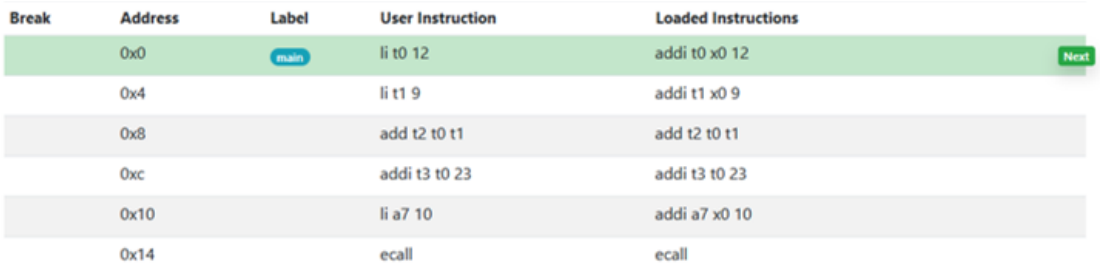

Figura 8: Programa cargado en memoria

En el panel de Registros, hay un botón *Memory* que permite ver el estado de la memoria una vez cargado el programa, tanto de datos (*Data*), como las instrucciones (*Text*) o la pila (*Stack*). En el caso del segmento de Texto, que es donde se guardan las instrucciones, se visualizan las direcciones que ocupan las instrucciones, su codificación en hexadecimal y la instrucción correspondiente (Fig. 9).

Una vez cargado el programa, podemos ejecutarlo paso a paso con el botón *Instr* o ejecutarlo de principio a fin con el botón *Run*. Para volver a ejecutarlo se emplea el botón *Reset*. Al ejecutarlo, se muestra la instrucción que se está ejecutando en azul (*Current*) y la instrucción siguiente en verde (*Next*), ver Fig. 10.

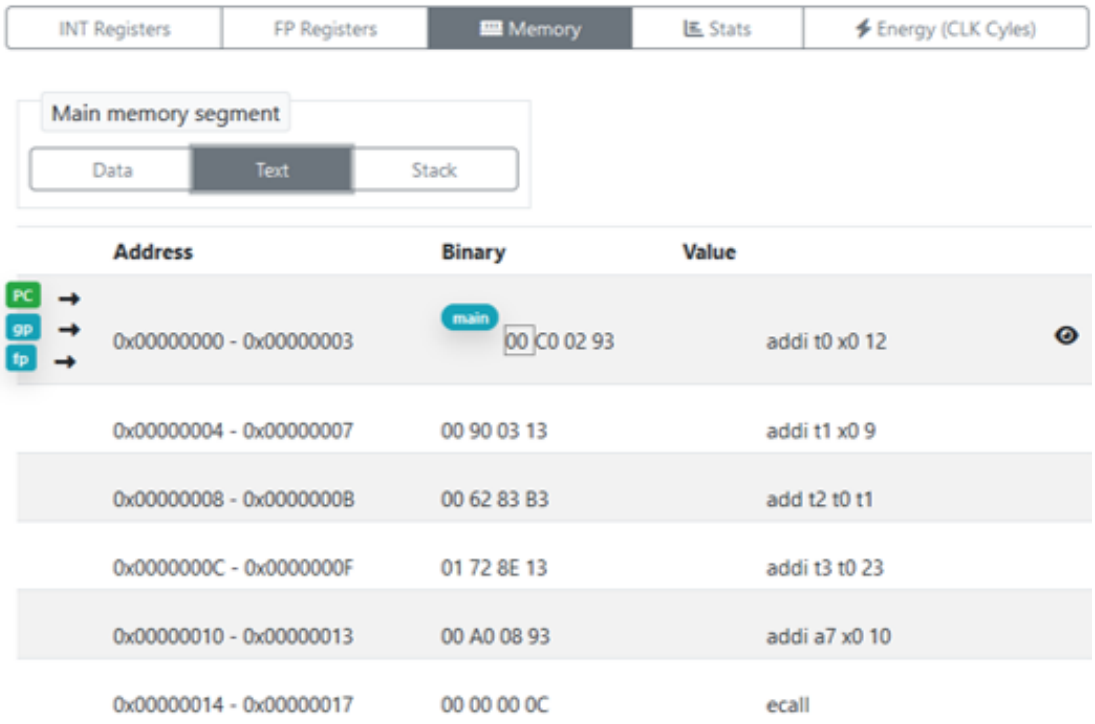

Figura 9: Estado del segmento de texto con el programa cargado

4. Los registros de la arquitectura RISC-V

Tiene el registro PC donde se almacena la dirección de la instrucción a ejecutar y 32 registros de 32 bits. El ABI de RISC-V (*Application Binary Interface* o Interfaz Binaria de Aplicaciones) da un nombre a cada registro (tabla 1).

Las extensiones opcionales RV32F y RV32D incluyen las instrucciones de coma flotante de precisión simple y doble. RISC-V sigue el estándar de coma flotante IEEE754-2008.

Para coma flotante, ambas extensiones RV32F y RV32D emplean registros adicionales (`f0-f31`). Así se mejora el rendimiento, duplicando la capacidad de los registros y el ancho de banda (más registros con el mismo número de bits para especificarlos).

5. Instrucciones aritmético-lógicas

5.1. Operaciones con registros

- **Suma:** `add x5, x6, x7` Hace la operación `x5 = x6 + x7`.

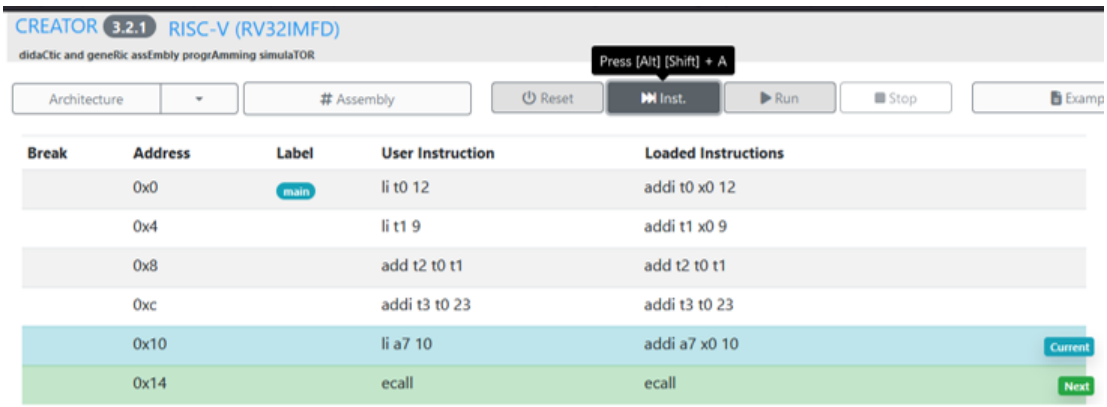

Figura 10: Programa ejecutándose

Registro (32 bits)	Nombre	Uso
`x0`	`zero`	Cableado a 0
`x1`	`ra`	Dirección de retorno
`x2`	`sp`	Puntero a pila
`x3`	`gp`	Puntero global
`x4`	`tp`	Puntero a hilo
`x5-x7`	`t0-t2`	Temporal
`x8`	`s0 / fp`	Registro preservado, puntero a frame
`x9`	`s1`	Registro preservado
`x10-x11`	`a0-a1`	Argumento de función / Valor de retorno
`x12-x17`	`a2-a7`	Argumento de función
`x18-x27`	`s2-s11`	Registro preservado
`x28-x31`	`t3-t6`	Temporal

Tabla 1: Registros de RISC-V

- **Resta:** `sub x5, x6, x7` Hace la operación `x5 = x6 - x7`.
- **And:** `and x5, x6, x7` Hace la operación `x5 = x6 & x7`.
- **Or:** `or x5, x6, x7` Hace la operación `x5 = x6 | x7`.
- **Xor:** `xor x5, x6, x7` Hace la operación `x5 = x6 ^ x7`.

5.2. Operaciones con inmediato

- **Suma:** `addi x5, x6, 15` Hace la operación `x5 = x6 + 15`.
- **And con inmediato:** `andi x5, x6, 15` Hace la operación `x5 = x6 & 15`.

- **Or con inmediato:** `ori x5, x6, 15` Hace la operación `x5 = x6 | 15`.
- **Xor con inmediato:** `xori x5, x6, 15` Hace la operación `x5 = x6 ^ 15`.

5.3. Instrucciones de desplazamiento

- **Desplazamiento lógico a la izquierda:** `sll x5, x6, x7` Hace la operación `x5 = x6 ´ x7`.
- **Desplazamiento lógico a la derecha:** `srl x5, x6, x7` Hace la operación `x5 = x6 ˇ x7`.
- **Desplazamiento aritmético a la derecha:** `sra x5, x6, x7` Hace la operación `x5 = x6 ˇ > x7`.
- **Desplazamiento lógico a la izquierda con inmediato:** `slli x5, x6, 15` Hace la operación `x5 = x6 ´ 15`.
- **Desplazamiento lógico a la derecha con inmediato:** `srli x5, x6, 15` Hace la operación `x5 = x6 ˇ 15`.
- **Desplazamiento aritmético a la derecha con inmediato:** `srai x5, x6, 15` Hace la operación `x5 = x6 ˇ > 15`.

6. Instrucciones de multiplicación y división

Están incluidas en el módulo RV32M. Son de tipo R.

- **Multiplicación:** `mul x5, x6, x7` Hace la operación `x5 = x6 * x7`. Se queda con los 32 bits de menos peso.
- **Multiplicación con signo:** `mulh x5, x6, x7` Hace la operación `x5 = x6 * x7`. Se queda con los 32 bits de más peso (los dos operandos son con signo).
- **Multiplicación sin signo:** `mulhu x5, x6, x7` Hace la operación `x5 = x6 * x7`. Se queda con los 32 bits de más peso (los dos operandos son sin signo).
- **Multiplicación con y sin signo:** `mulhsu x5, x6, x7` Hace la operación `x5 = x6 * x7`. Se queda con los 32 bits de más peso (un operando es con signo y otro sin signo).
- **División sin signo:** `divu x5, x6, x7` Hace la operación `x5 = x6 / x7`.

- **División con signo:** `div x5, x6, x7` Hace la operación `x5 = x6 / x7`.
- **Resto sin signo:** `remu x5, x6, x7` Hace la operación `x5 = x6 % x7`.
- **Resto:** `rem x5, x6, x7` Hace la operación `x5 = x6 % x7`.

7. Ejemplo

Un ejemplo de un programa en ensamblador en el simulador CREATOR es el siguiente:

```
1
2   .text # comienzo del segmento de texto (instrucciones)
3   .globl main # la etiqueta main es global
4
5 main: # Comienza la función main
6   li t0, 12 # Pseudoinstrucción. Asigna 12 al registro t0
7   li t1, 9 # Pseudoinstrucción. Asigna 9 al registro t1
8
9   add t2, t0, t1     # t2 = t0 + t1
10  addi t3, t0, 23   # t3 = t0 + 23
11
12  li a7, 10 # código de llamada exit
13  ecall # llamada al sistema
```

Una **pseudoinstrucción** es una instrucción que no existe en el repertorio del procesador pero el ensamblador la traduce a una que sí existe.

Abre el simulador, haz clic en *#Assembly* y pega el código anterior (puedes crear un nuevo archivo *File* → *New*).

Guarda el archivo (extension `.s`) y compílalo (*Compile/Linked*). Si todo va bien, saldrá un mensaje de que la compilación se ha realizado con éxito.

Haz clic en el botón *Simulator* y veremos el programa cargado. Se puede visualizar el segmento de texto si hacemos clic en el botón *Memory* → *Text*.

Ejecuta el programa paso a paso para ver cómo van cambiando de valor los registros.

Se pueden poner **puntos de ruptura** haciendo clic en la instrucción correspondiente en la columna *Break*. Si le das a ejecutar se para en esa instrucción y se pueden ver los valores que toman los registros en ese punto.

8. Ejercicio

Realiza los siguientes ejercicios teniendo en cuenta que la estructura de un programa es la siguiente:

```
1
2   .text # comienzo del segmento de texto (instrucciones)
3   .globl main # la etiqueta main es global
4
5 main: # Comienza la función main
6
7   # Aquí van las instrucciones
8
9   li a7, 10 # código de llamada exit
10  ecall # llamada al sistema
```

8.1. Ejercicio

Intercambia los valores de los registros `x1` y `x2` sin usar registros extra. En C hacemos:

```
aux=a;
a=b;
b=aux;
```

Se puede hacer sin usar registros extra con la instrucción `xor`. En lenguaje ensamblador del RISC-V:

- `xor x1, x1, x2` → `x1 = x1 ^ x2`
- `xor x2, x1, x2` → `x2 = (x1 ^ x2)^ x2 = x1` puesto que `x2 ^ x2 = 0`
- `xor x1, x1, x2` → `x1 = (x1 ^ x2) ^ x1 = x2` puesto que por la propiedad conmutativa y asociativa de la operación `xor` se tiene que `(x1 ^ x2) ^ x1 = (x1 ^ x1) ^ x2 = x2`

Escribe un programa que le asigne al registro `t0` el valor 15 y al registro `t1` el valor 11 e intercambie su valor usando la instrucción `xor`. Ejecútalo paso a paso para ver cómo cambia de valor.

8.2. Ejercicio

Crea un programa llamado que haga lo siguiente:

- Inicialice los registros `t0` y `t1` con el valor 64 y 2, respectivamente.
- Calcule el desplazamiento lógico a la derecha del registro `t0` tantas veces como indica `t1` y lo guarde en `t2`.
- Calcule el resto de dividir 17 entre 3 y lo guarde en `t3`.
- Salga del programa.