uc3m | Universidad Carlos III de Madrid

Grupo ARCOS

XIII Seminario de Invierno CAPAP-H, Almería, 1, 2 y 3 de febrero de 2023

CREATOR como herramienta docente para la enseñanza de la programación en ensamblador con RISC V

Félix García Carballeira

felix.garcia@uc3m.es

Material

https://github.com/fgcarbal/Creator/

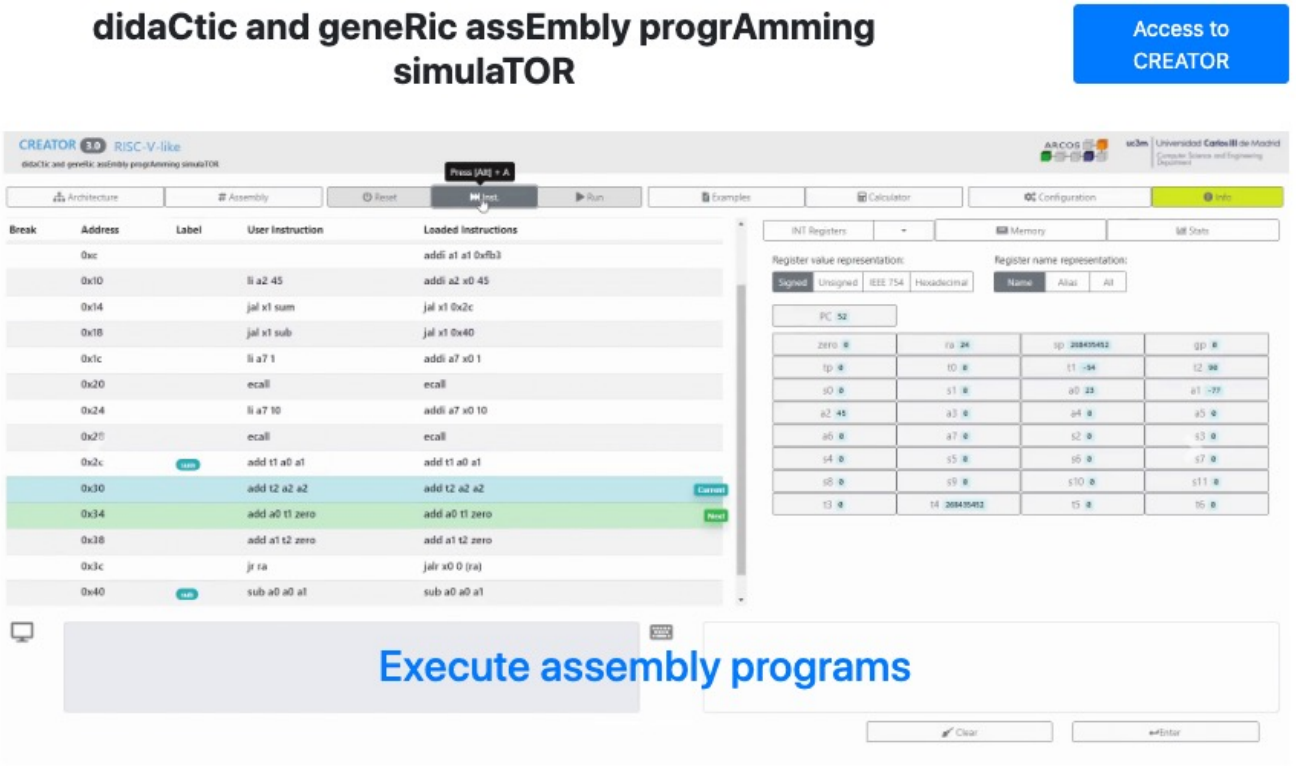
Motivación de Creator
didaCtic and geneRic assEmbly prograMming simulaTOR

- Simulador didáctico para la enseñanza de la programación en ensamblador
  - Centrado en los estudiantes y profesores
- Multiplataforma
  - Ejecución en web sin servidor (sobremesa, tablets y móviles)
- Entorno integrado (edición, compilación y simulación de programas)
- Posibilidad de definir y trabajar con diferentes arquitecturas y lenguajes ensamblador
  - Características básicas (nº de bits, registros, …)
  - Instrucciones
  - Pseudoinstrucciones
  - Directivas

Creator desde el punto de vista docente

- Facilidades para entender:
  - La representación de datos e instrucciones
  - La diferencia entre instrucciones y pseudoinstrucciones
  - La carga de un programa en memoria
  - El flujo de ejecución de un programa en ensamblador conociendo en todo momento la instrucción en curso y la siguiente (útil en bucles)
  - El convenio de paso de parámetros y uso de pila con alertas cuando no se respeta
  - El concepto de biblioteca de funciones y su uso

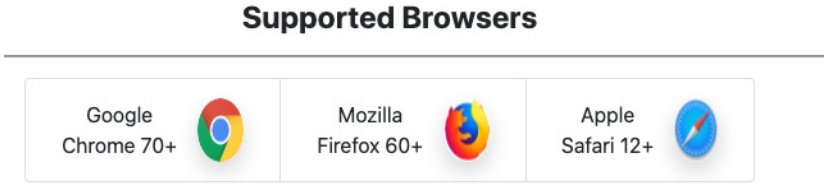
CREATOR



https://creatorsim.github.io/

Características

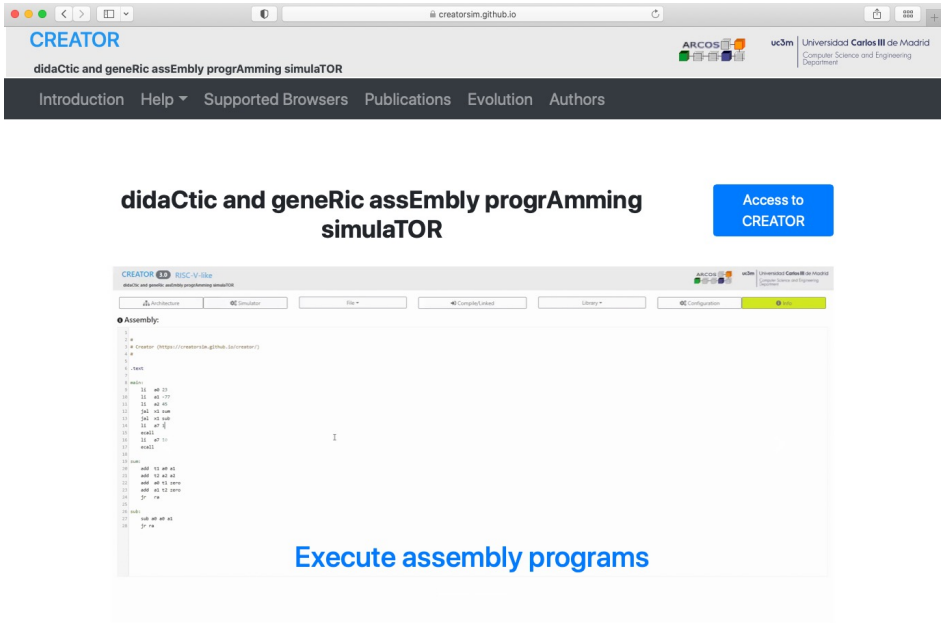
- Permite describir las características de una arquitectura y su juego de instrucciones
  - Actualmente: MIPS-32, RISC-V (RV32IMFD)
- Editar y compilar programas en ensamblador del juego de instrucciones elegido
- Ejecutar/depurar programas en ensamblador en un mismo entorno
- Obtener estadísticas sobre los programas ejecutados
- Ejecución en navegador



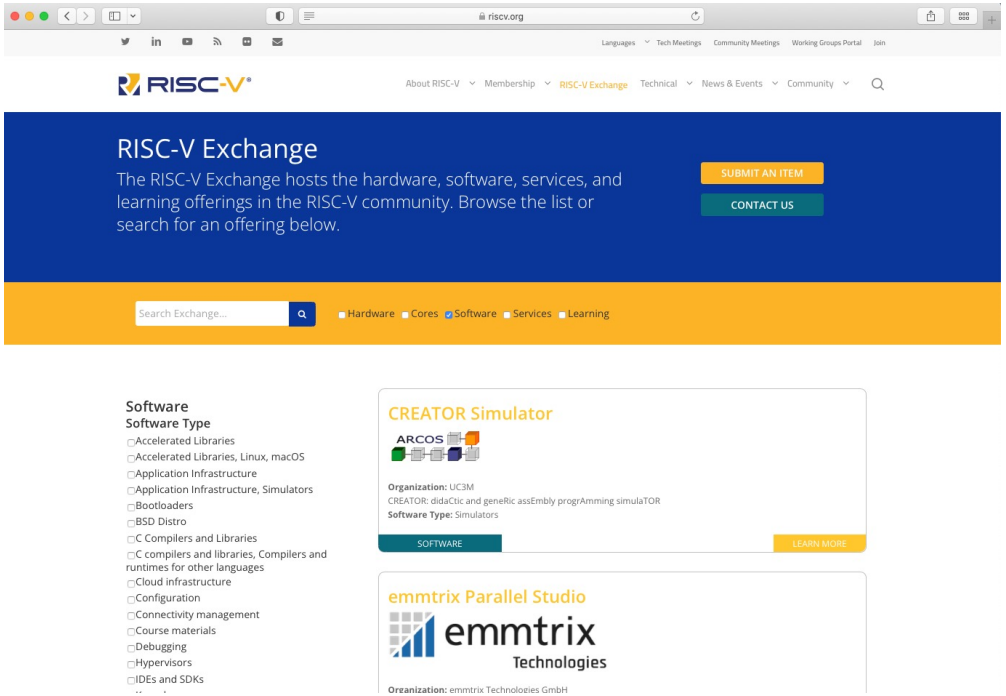
Contenido: empleo de CREATOR con RISC-V

- Juego de instrucciones soportado
- Visión del estudiante:
  - Características del entorno
  - Edición y compilación de programas
  - Ejecución y depuración de programas
  - Bibliotecas de funciones
  - Facilidades para entender el empleo de funciones y uso de pila
- Visión del profesor:
  - Soporte a la corrección de prácticas
  - Soporte a la creación de material didáctico
  - Capacidades para extender el juego de instrucciones y crear nuevas arquitecturas

Disponibilidad



https://creatorsim.github.io



https://riscv.org/exchange

Juego de instrucciones soportado (RV32IMFD)

98 instrucciones y pseudoinstrucciones

Guía de referencia

- Transferencia de datos: `li, mv, lui`
- Aritméticas y lógicas sobre registros de enteros: `addi, add, and`, …
- Aritméticas sobre números en coma flotante (float y double): `fadd.s, fmul.d`, …
- Instrucciones de salto (registros enteros): `beq, bne`, …
- Instrucciones de comparación (enteros y coma flotante): `slt, feq.s`, …
- Instrucciones de transferencia entre registros enteros y coma flotante: `fmv.w.x`,
- Llamadas a funciones y llamadas al sistema: `jal, jr, ecall`
- Acceso a memoria (enteros y coma flotante): `lb, lw, flw, fsd`,….
- Operaciones de conversión (enteros y coma flotante): `fcvt.w.s`, …
- Otras:
  - Clasificación de coma flotante: `fclass.s, fclass.d`
  - Contador de ciclos: `rdcyle`

Registros

Integer Registers	
Register Name	**Usage**
zero	Constant 0
ra	Return address (routines/functions)
sp	Stack pointer
gp	Global pointer
tp	Thread pointer
t0..t6	Temporary (NOT preserved across calls)
s0..s11	Saved temporary (preserved across calls)
a0, a1	Arguments for functions / return value
a2..a7	Arguments for functions
Floating-point registers	
ft0..ft11	Temporary (NOT preserved across calls)
fs0..fs11	Saved temporary (preserved across calls)
fa0, fa1	Arguments for functions / return value
fa2..fa7	Arguments for functions

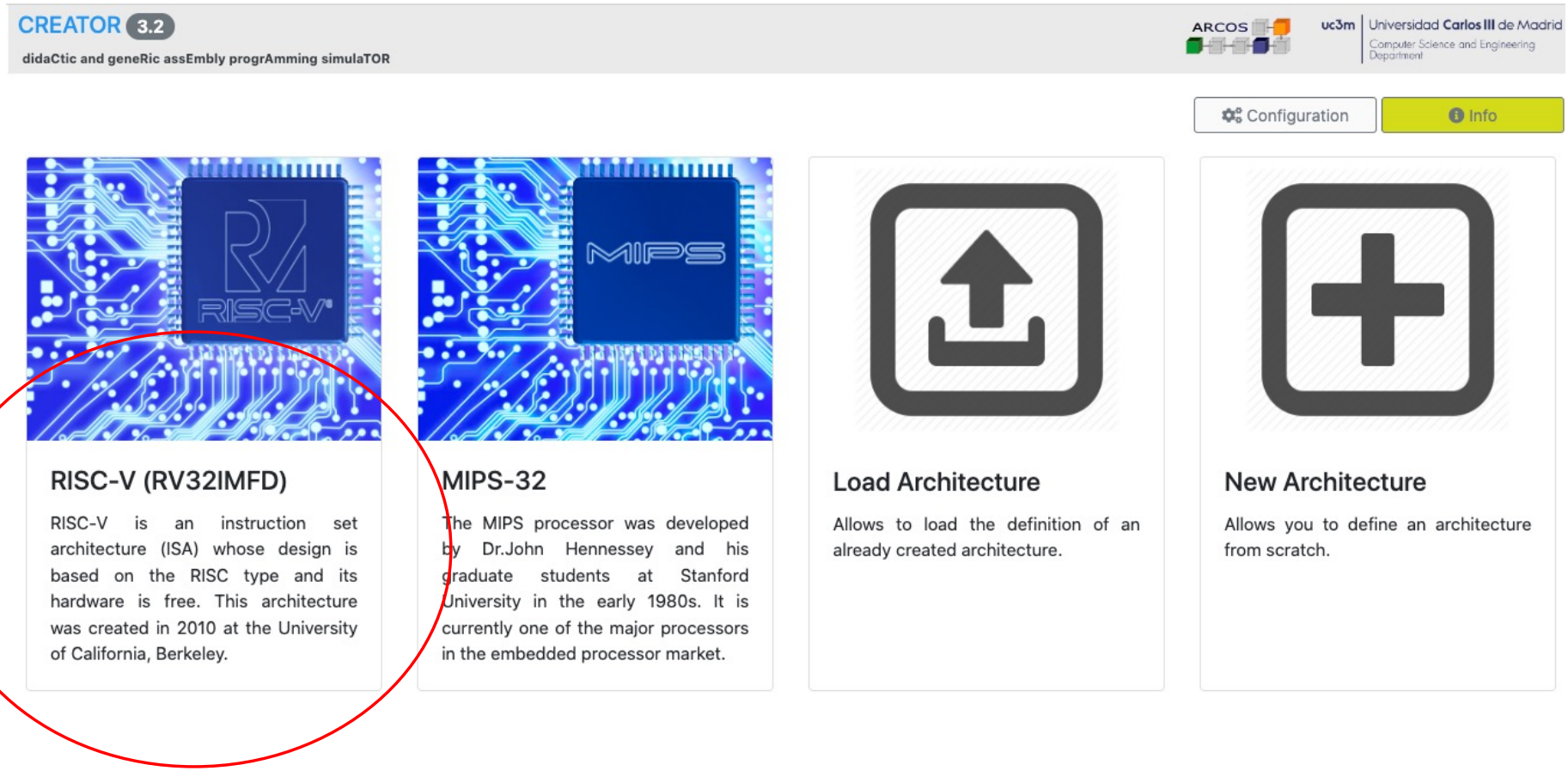
Llamadas al sistema

System Calls (ecall)			
Service	**Call Code (a7)**	**Arguments**	**Result**
Print_int	1	a0 = integer	
Print_float	2	fa0 = float	
Print_double	3	fa0 = double	
Print_string	4	a0 = string addr	
Read_int	5		Integer in a0
Read_float	6		Float in fa0
Read_double	7		Double in fa0
Read_string	8	a0 = string addr a1 = length	
Sbrk	9	a0 = length	Address in a0
Exit	10		
Print_char	11	a0 = ASCII code	
Read_char	12		Char in a0

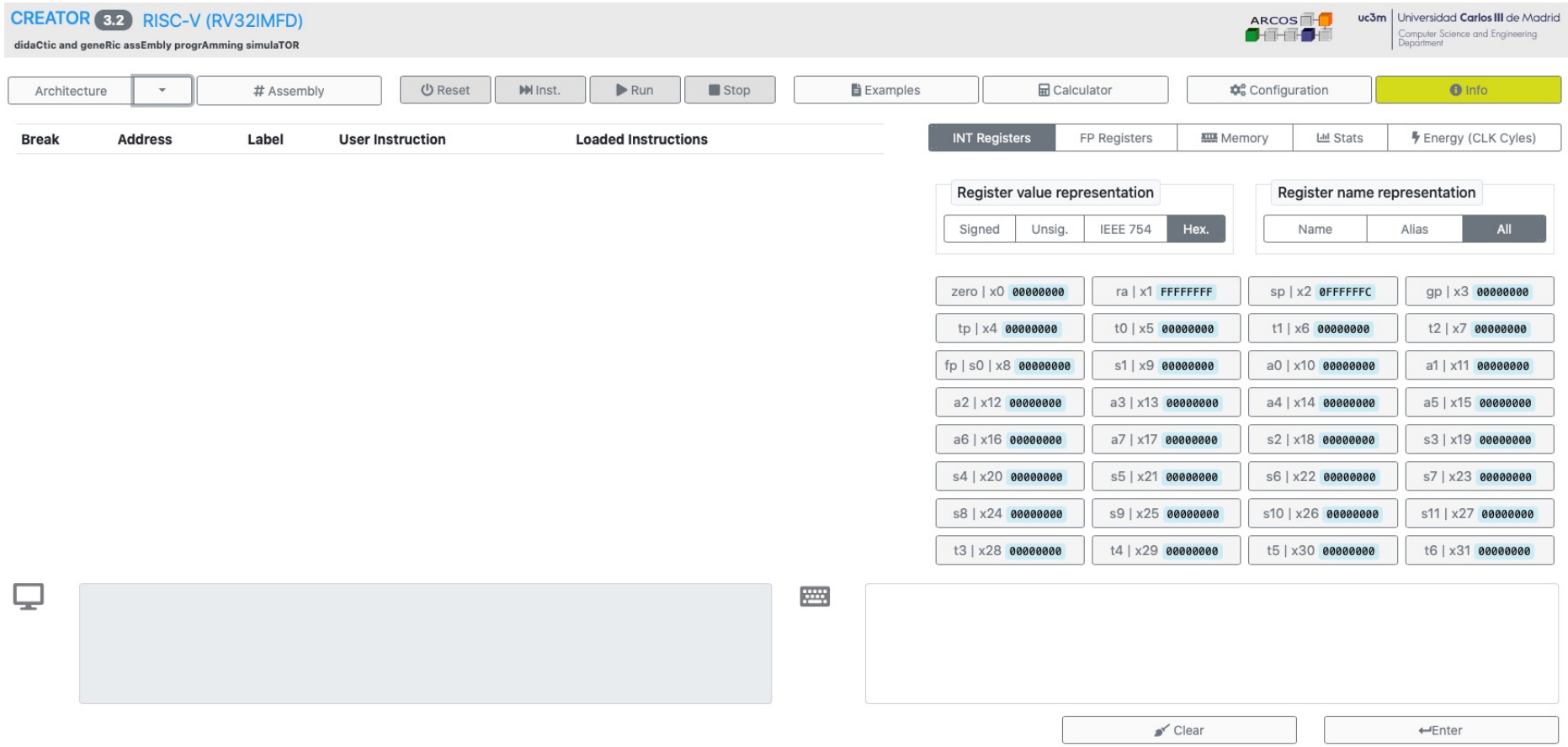
Directivas soportadas

Directivas	Uso
.data	Siguientes elementos van al segmento de dato
.text	Siguientes elementos van al segmento de código
.ascii *"tira de caracteres"*	Almacena cadena caracteres NO terminada en carácter nulo
.string *"tira de caracteres"*	Almacena cadena caracteres terminada en carácter nulo
.byte 1, 2, 3	Almacena bytes en memoria consecutivamente
.half 300, 301, 302	Almacena medias palabras en memoria consecutivamente
.word 800000, 800001	Almacena palabras en memoria consecutivamente
.float 1.23, 2.13	Almacena float en memoria consecutivamente
.double 3.0e21	Almacena double en memoria consecutivamente
.zero 10	Reserva un espacio de 10 bytes en el segmento actual
.align *n*	Alinea el siguiente dato en un límite de 2^n

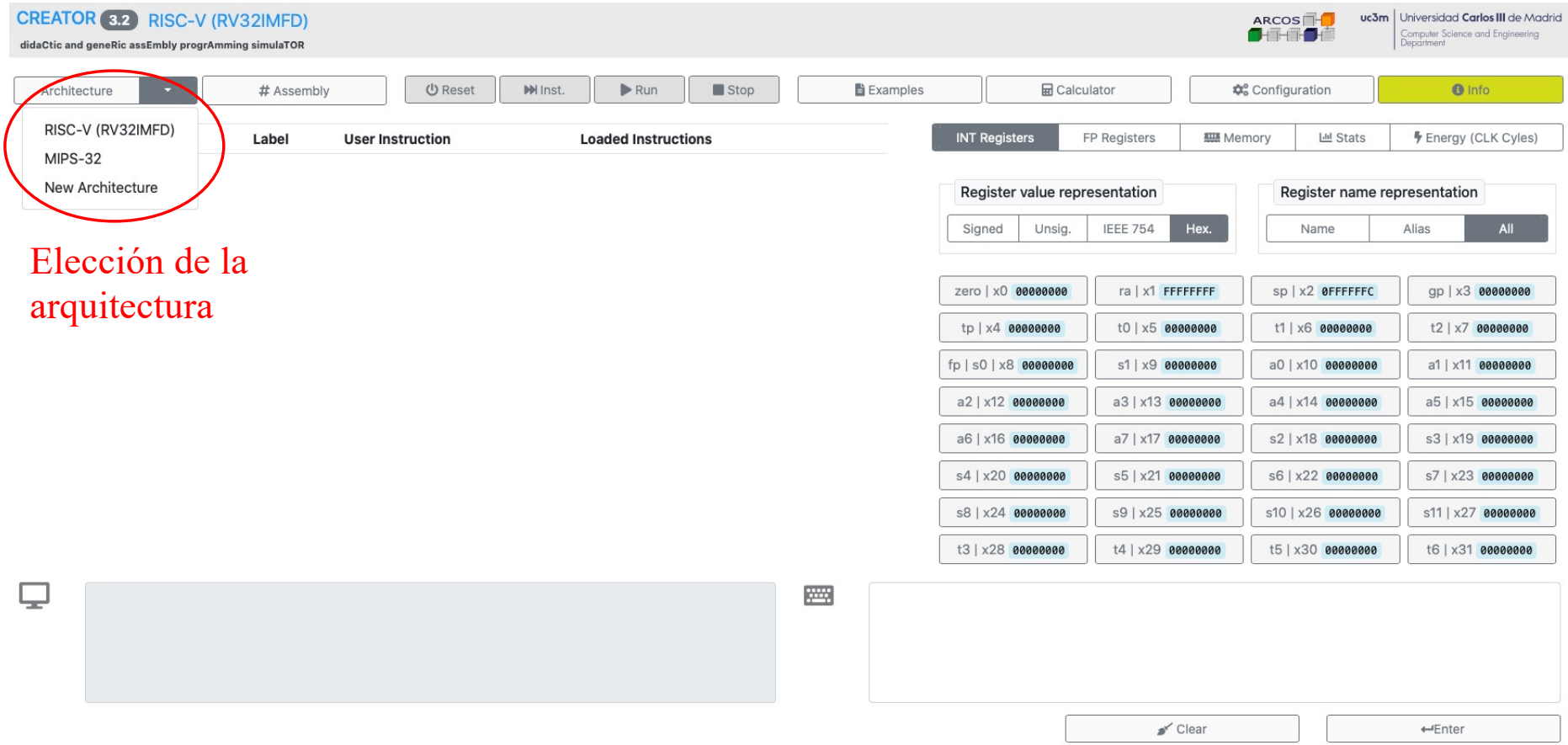
CREATOR (RISC-V)



Pantalla inicial

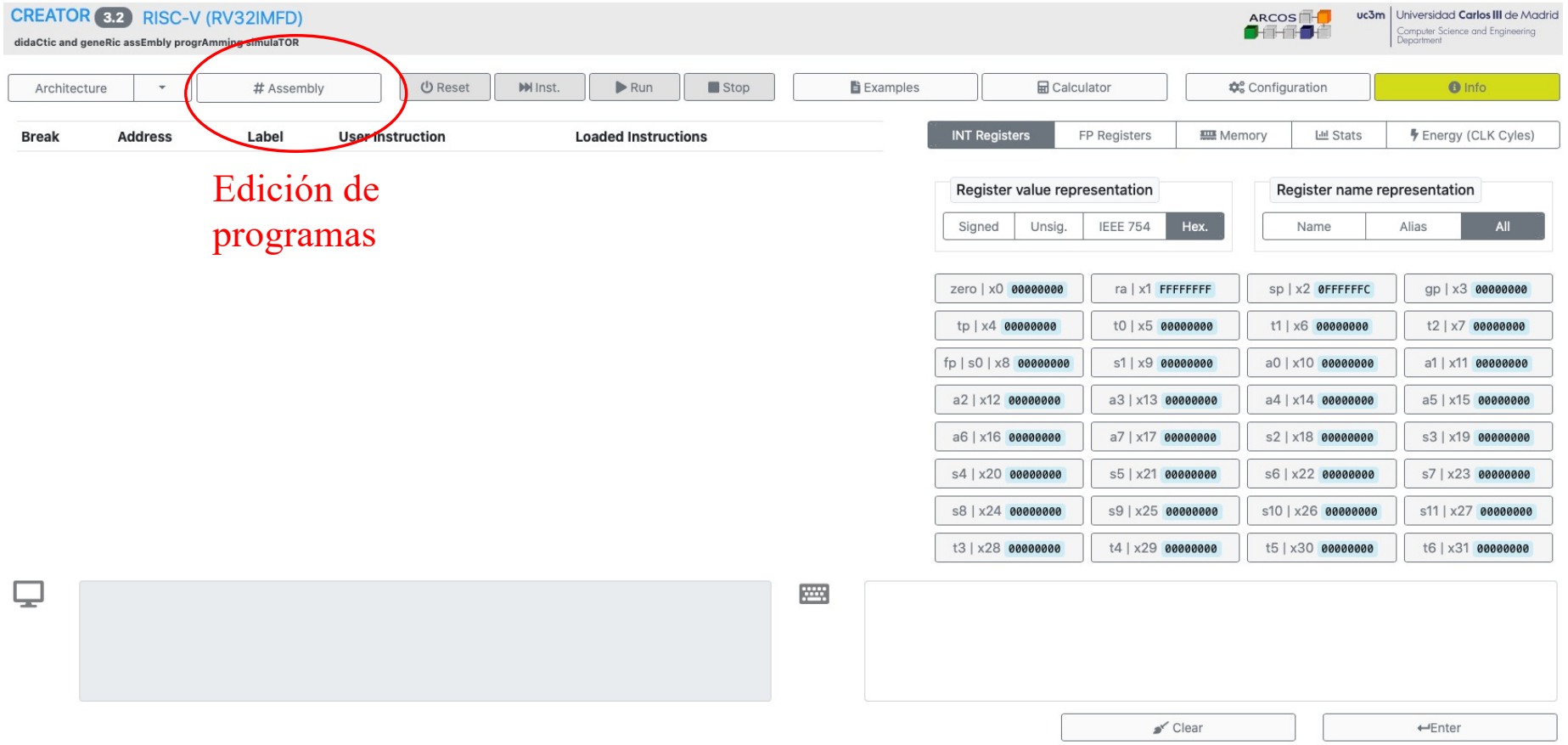


Elección de la arquitectura



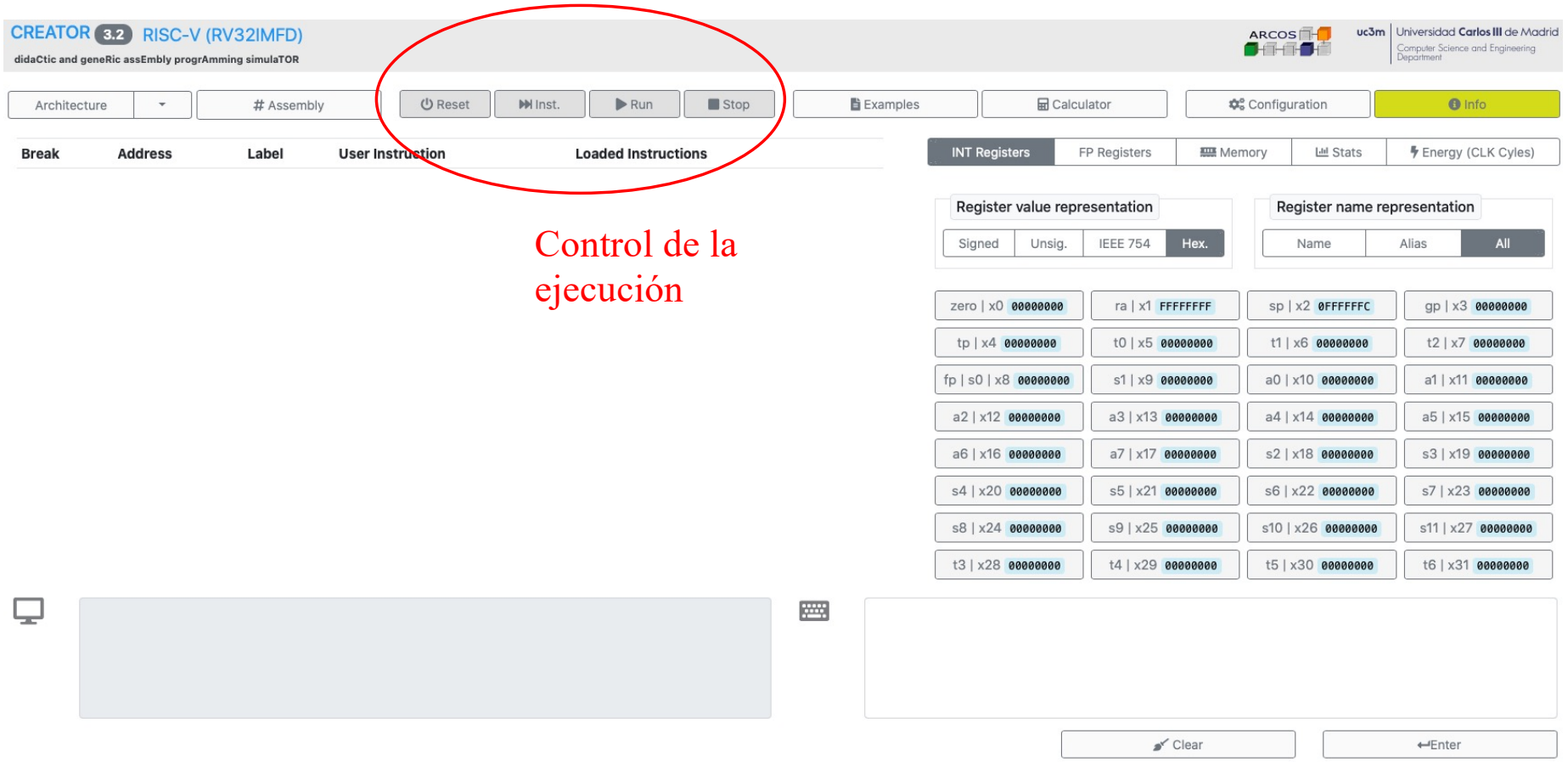
Elección de la arquitectura

Edición de programas



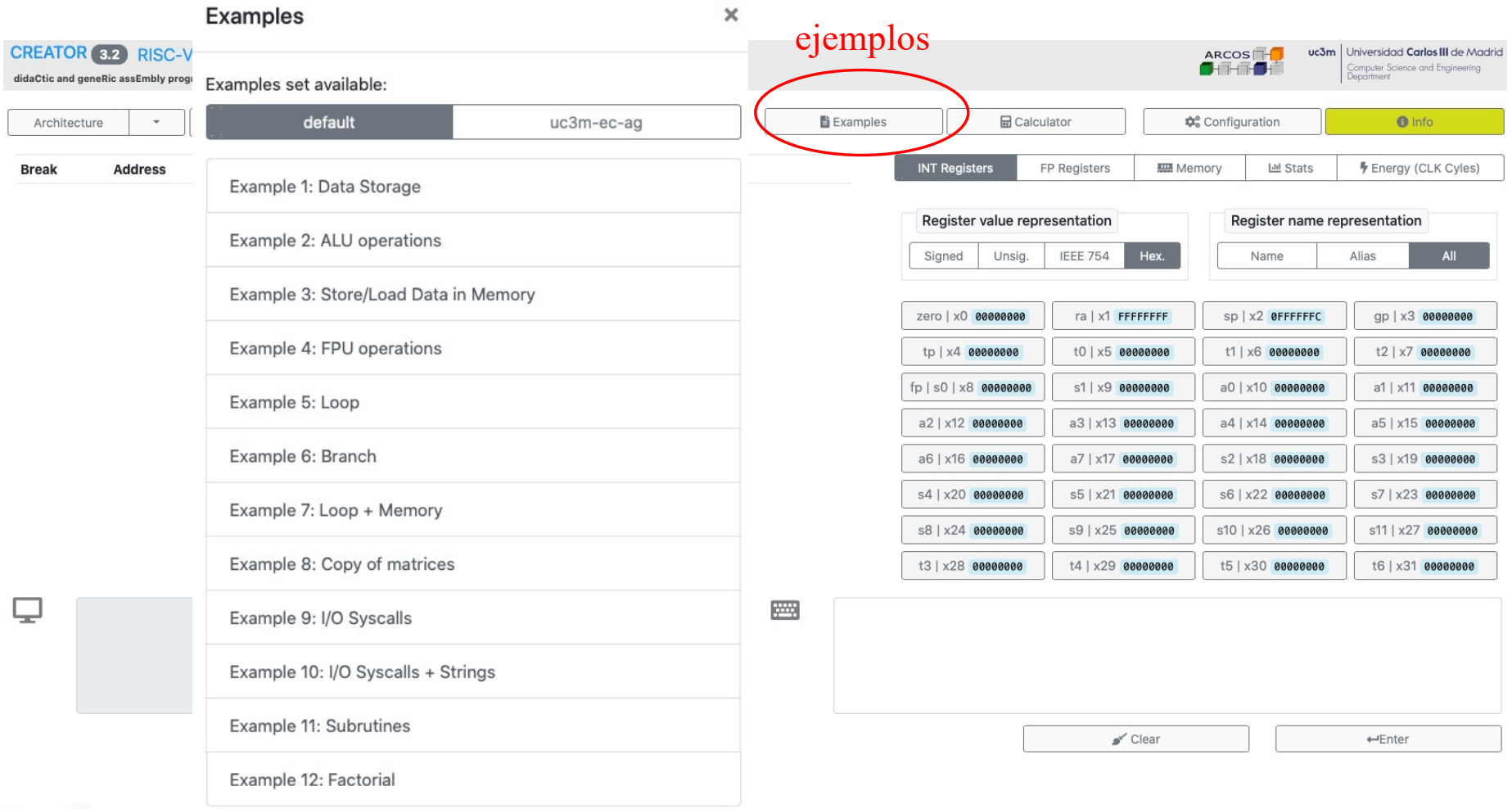
Edición de programas

Control de la ejecución

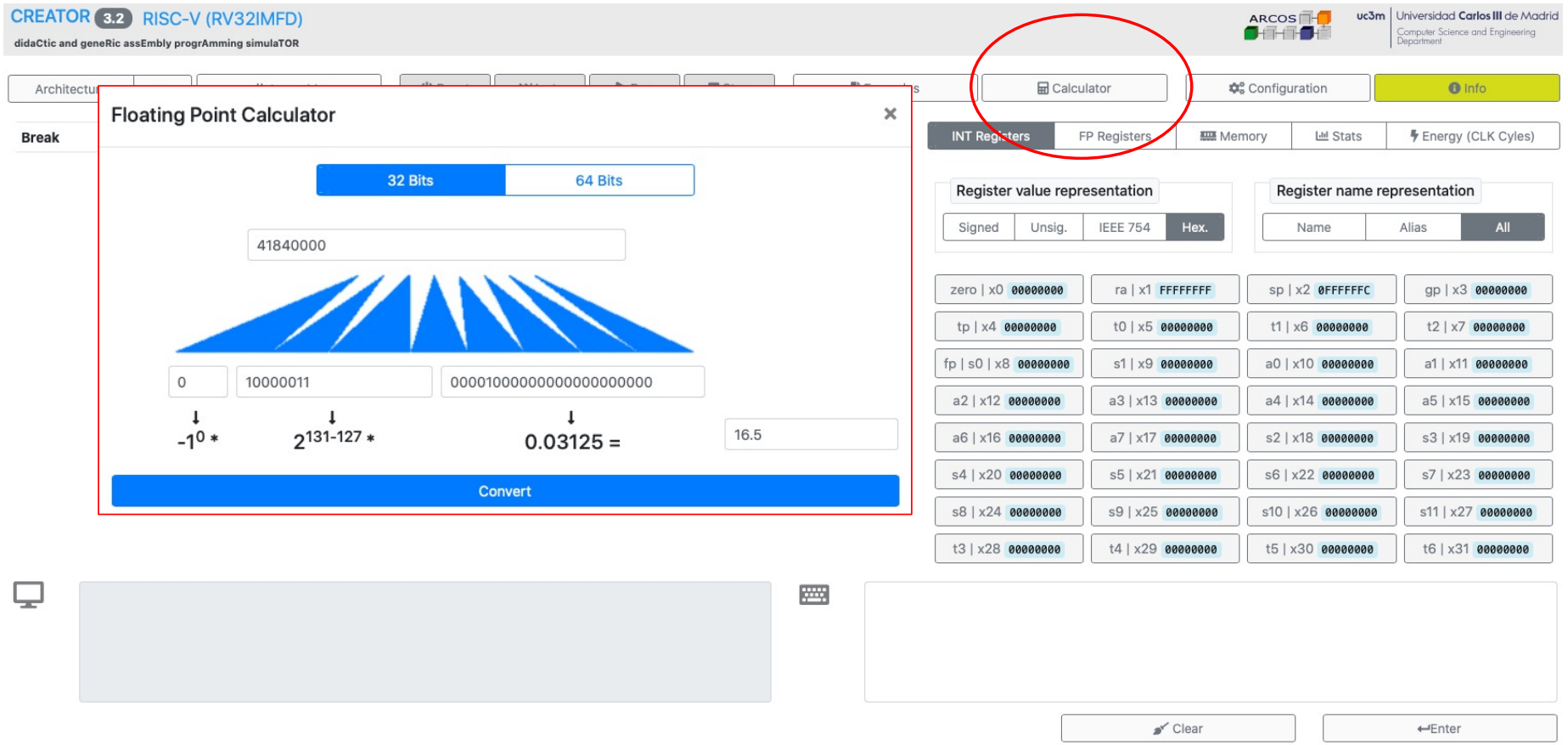


Control de la ejecución

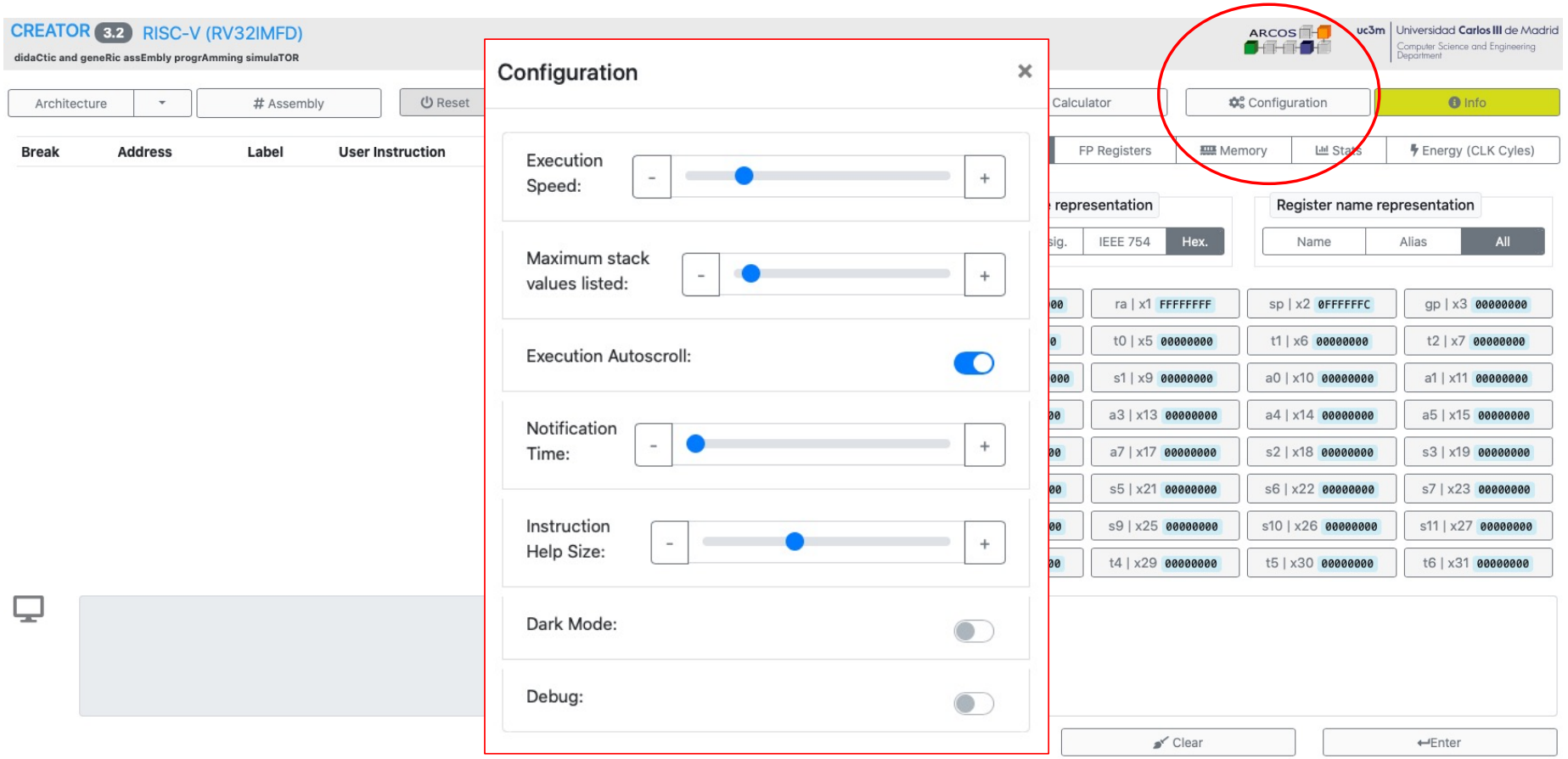
Ejemplos de programas en ensamblador



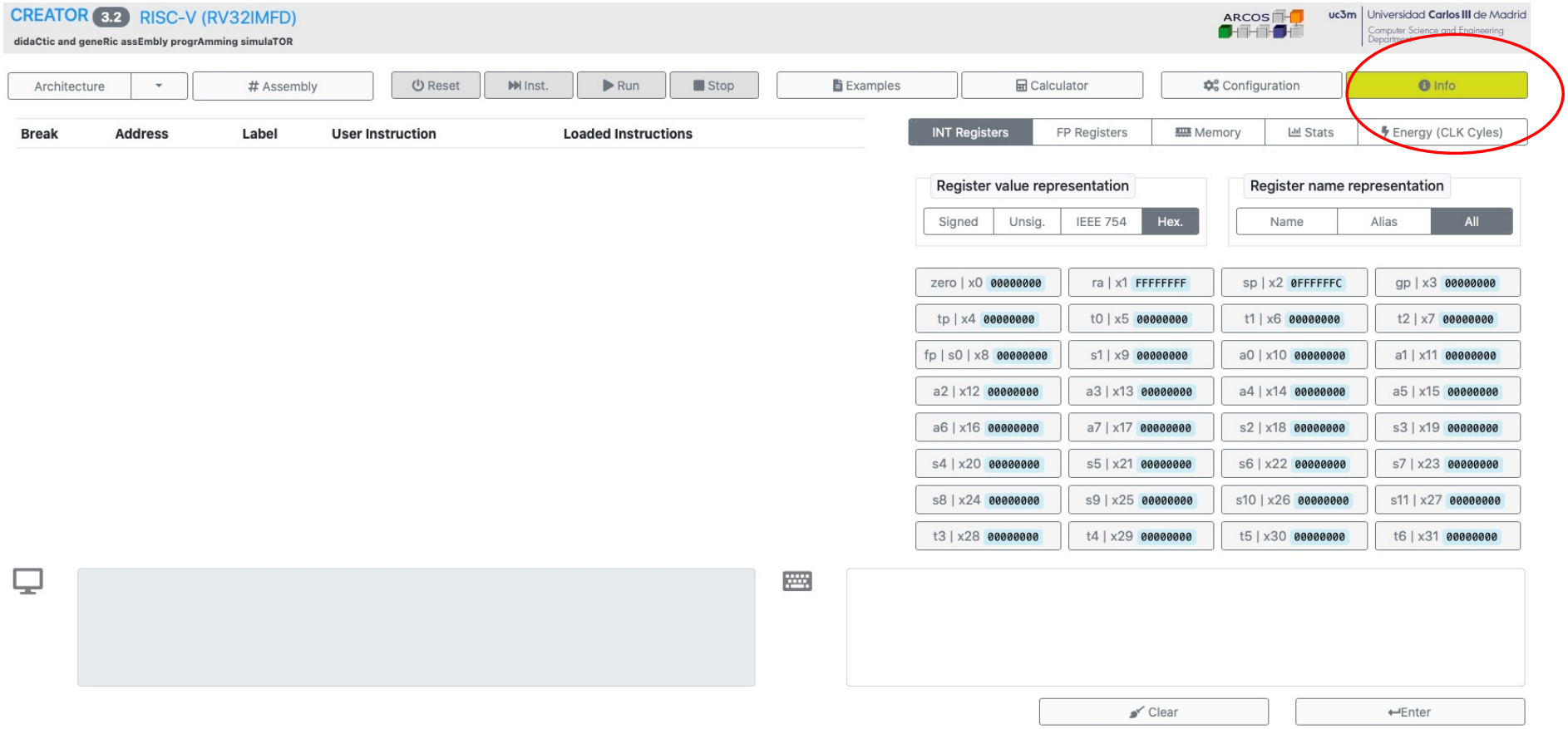
Calculadora de números en coma flotante



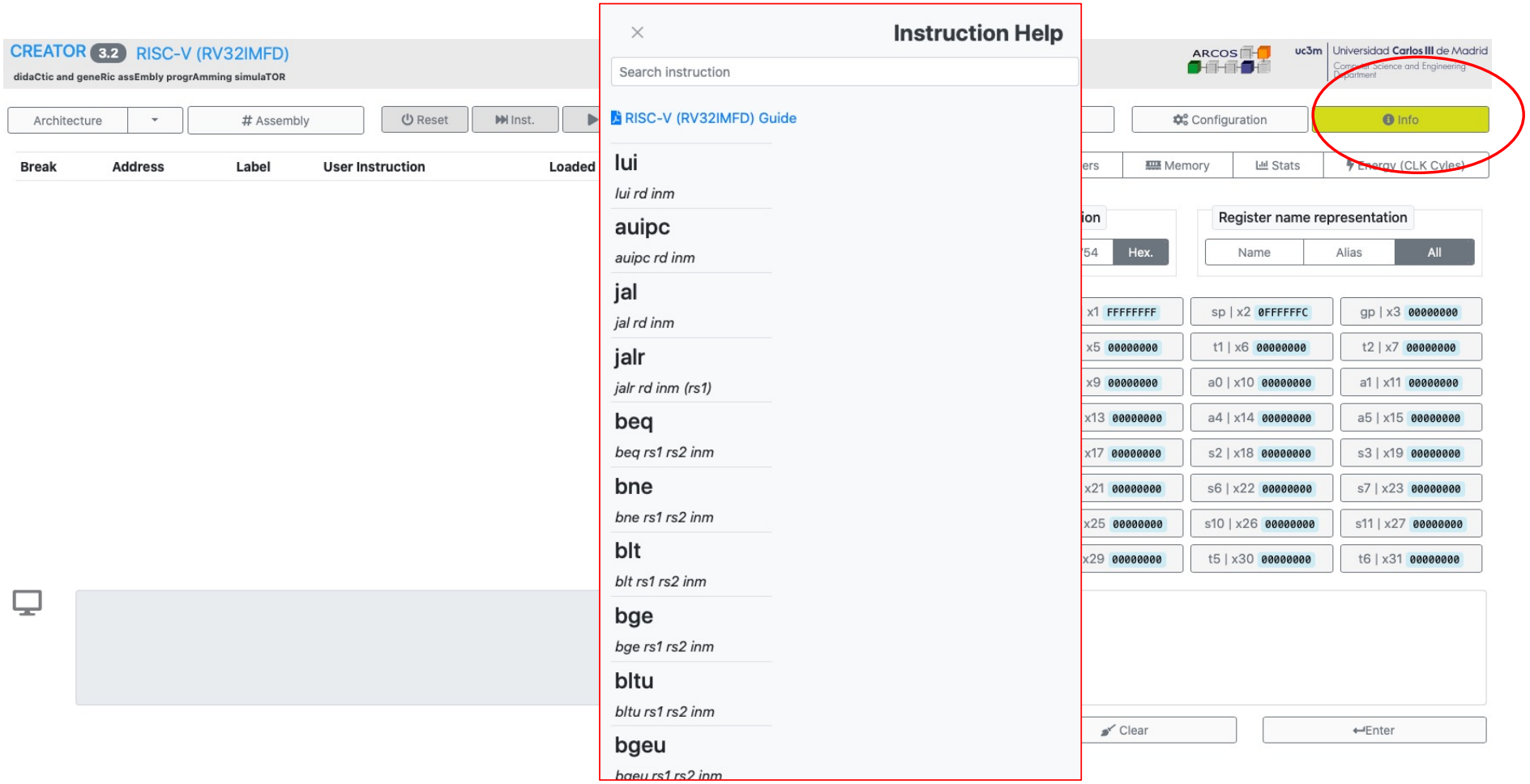
Configuración



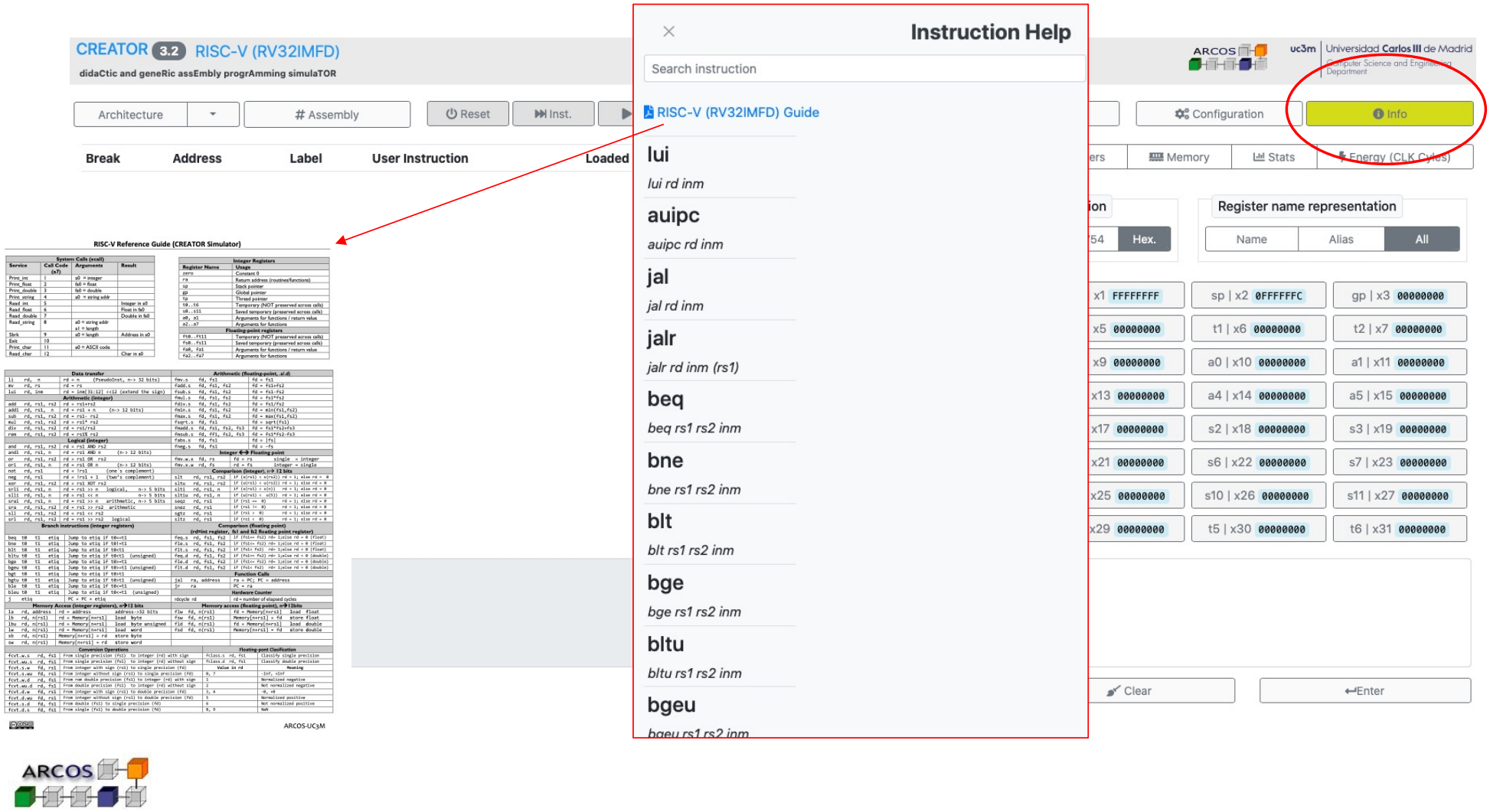
Configuración



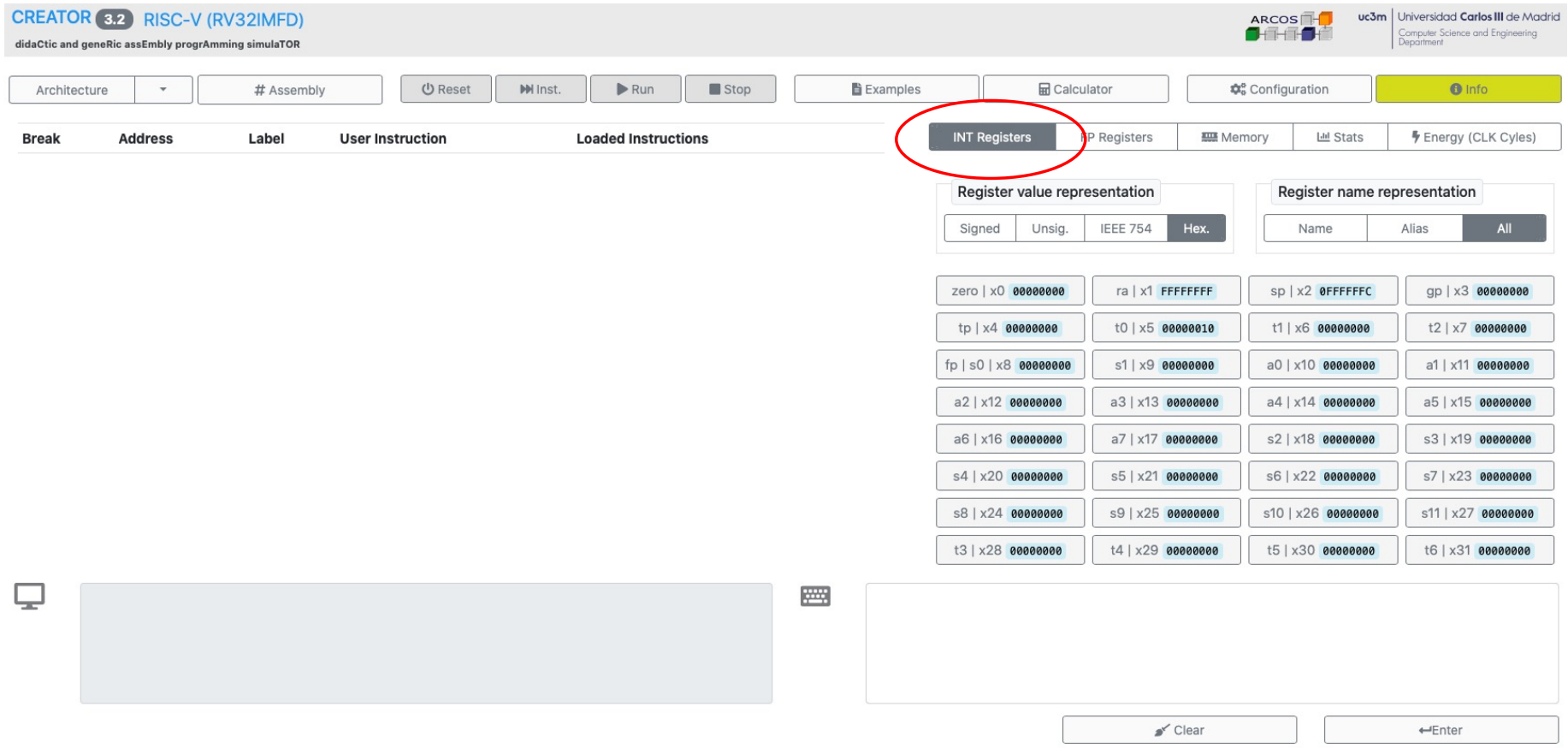
Configuración



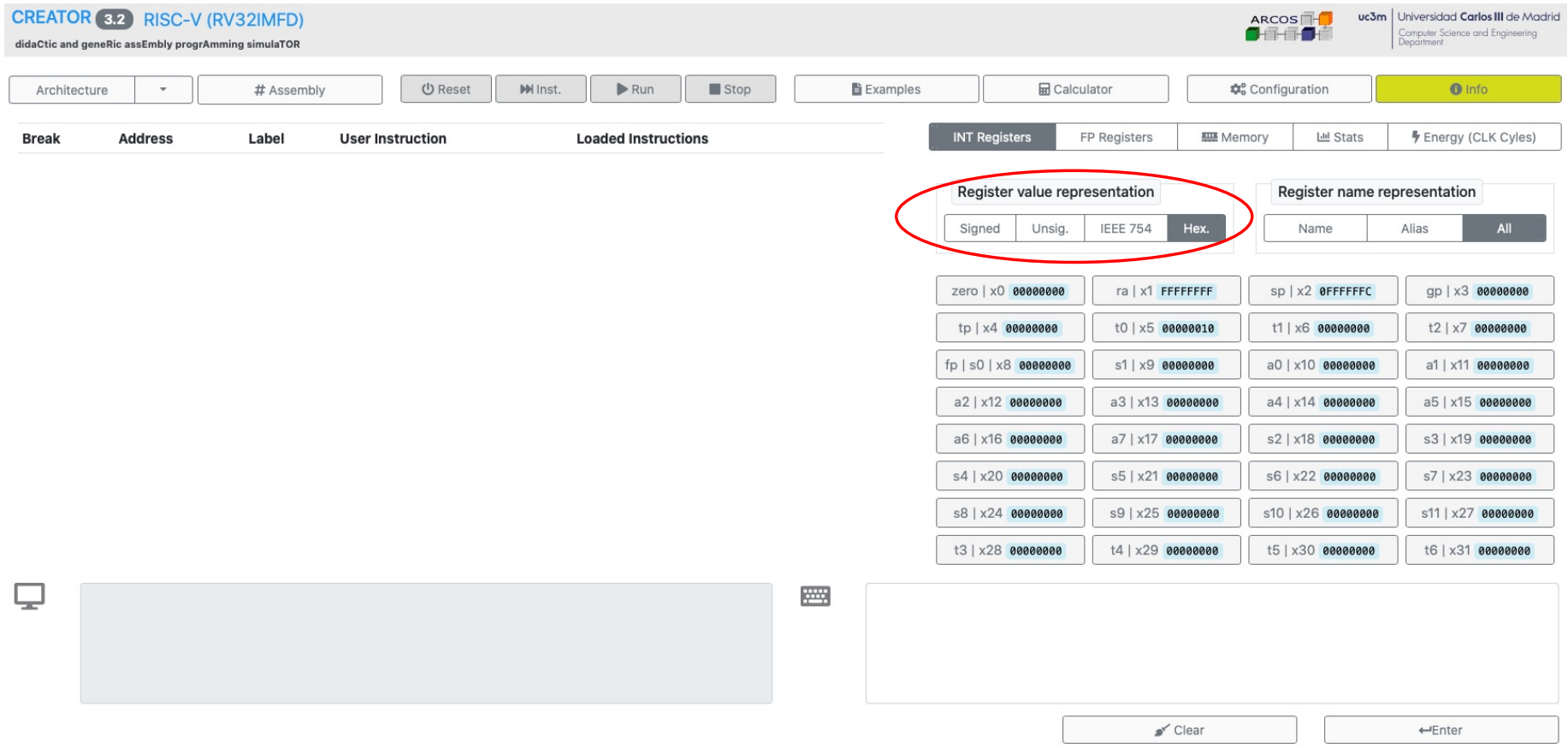
Configuración



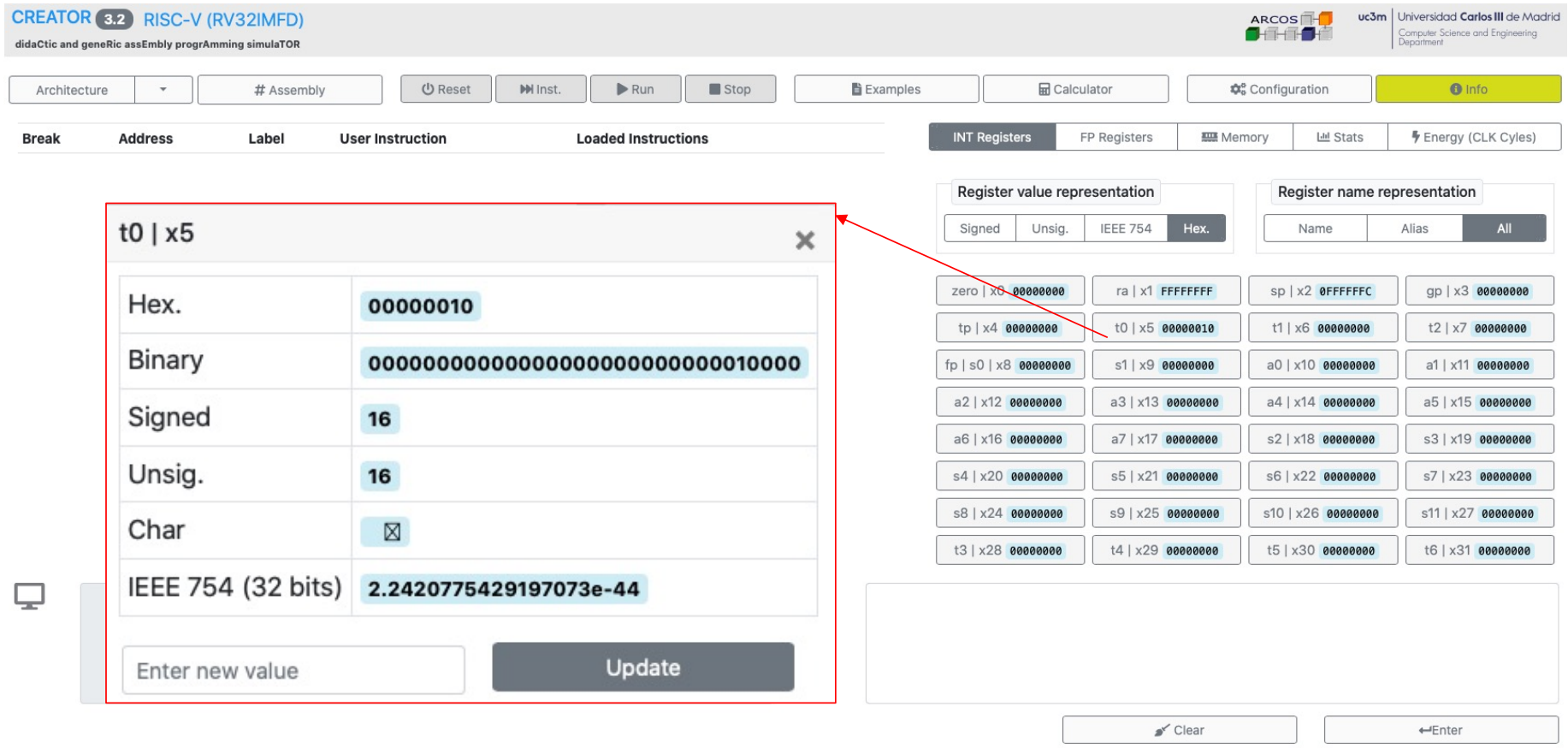
Registros enteros



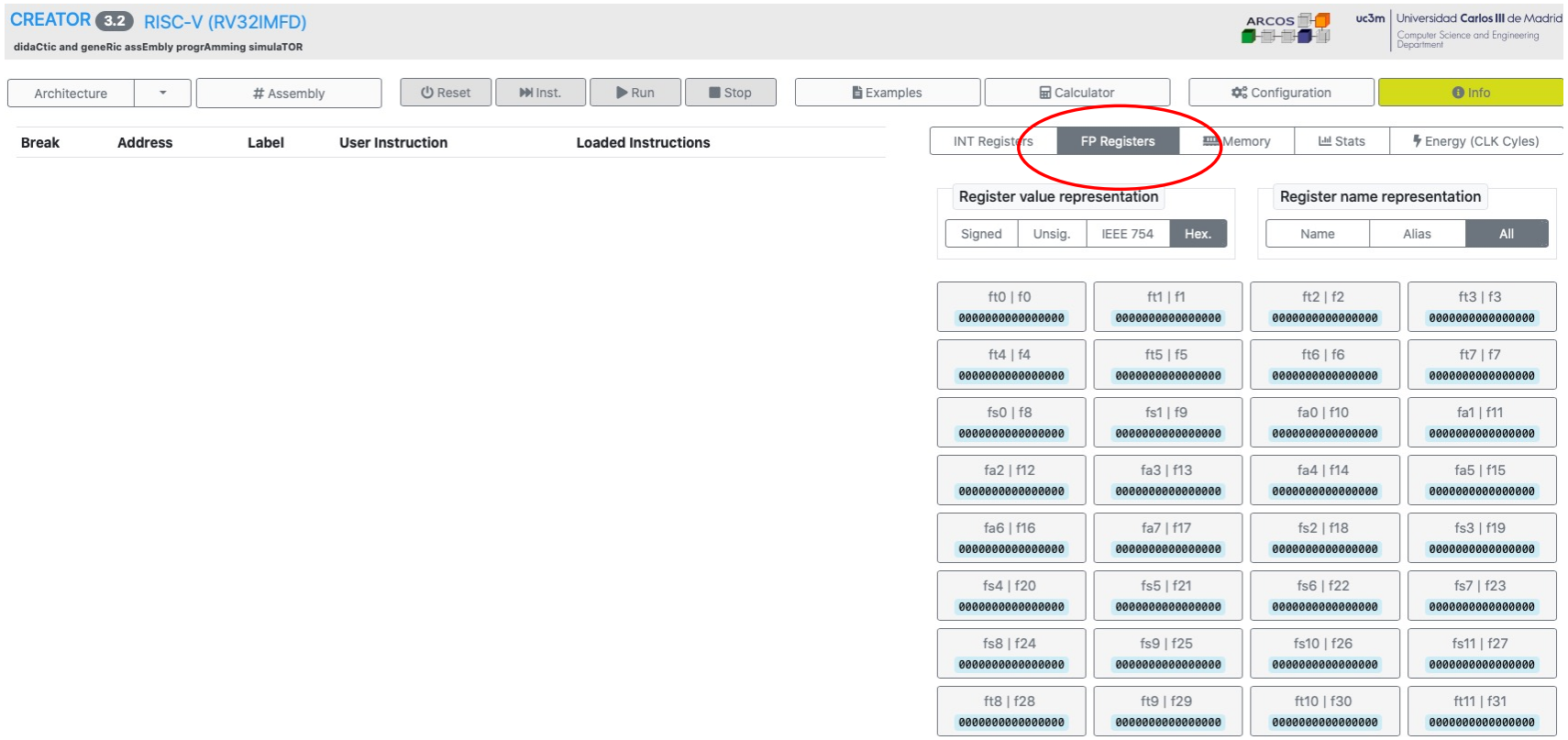
Registros enteros



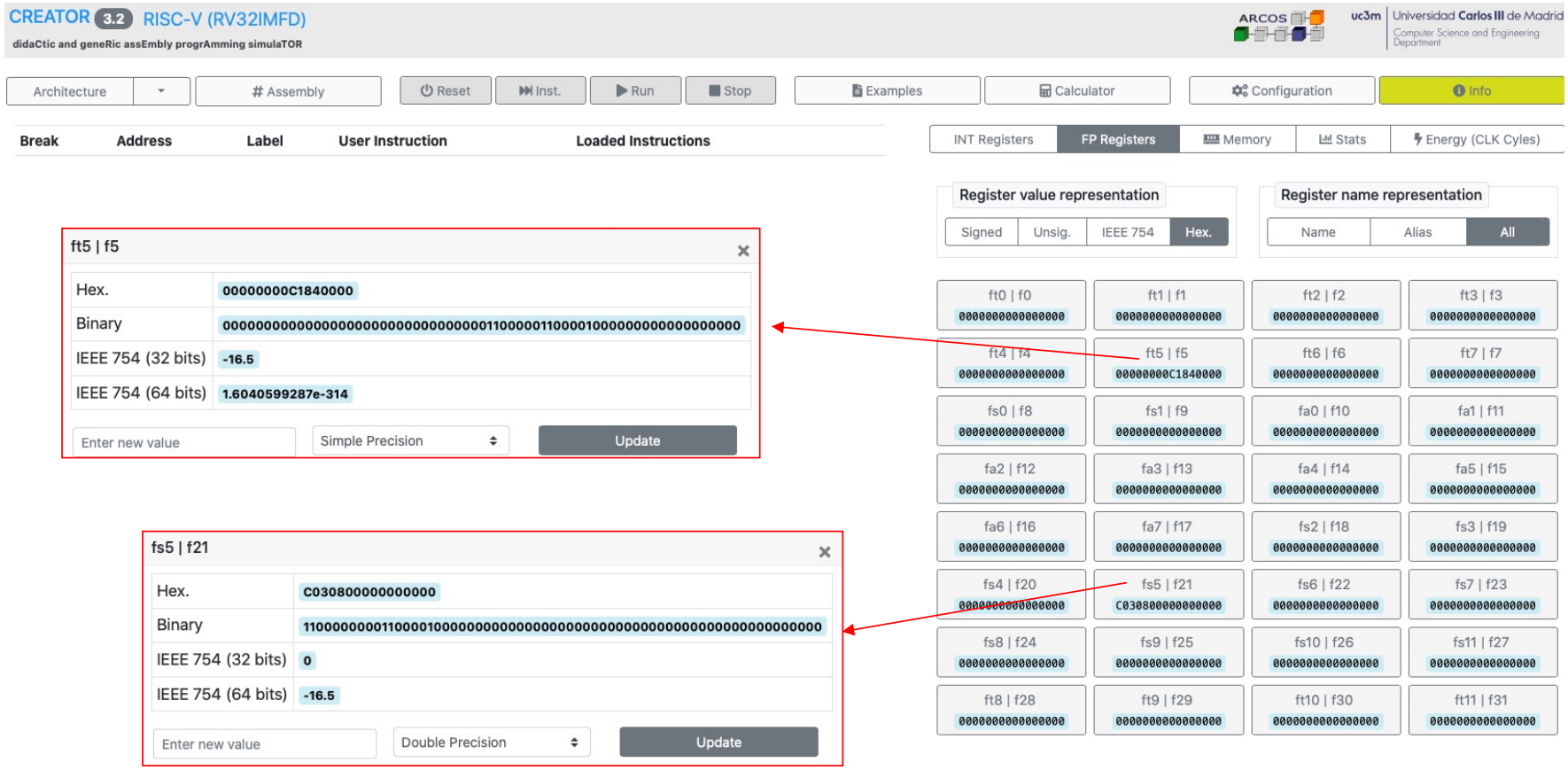
Registros enteros



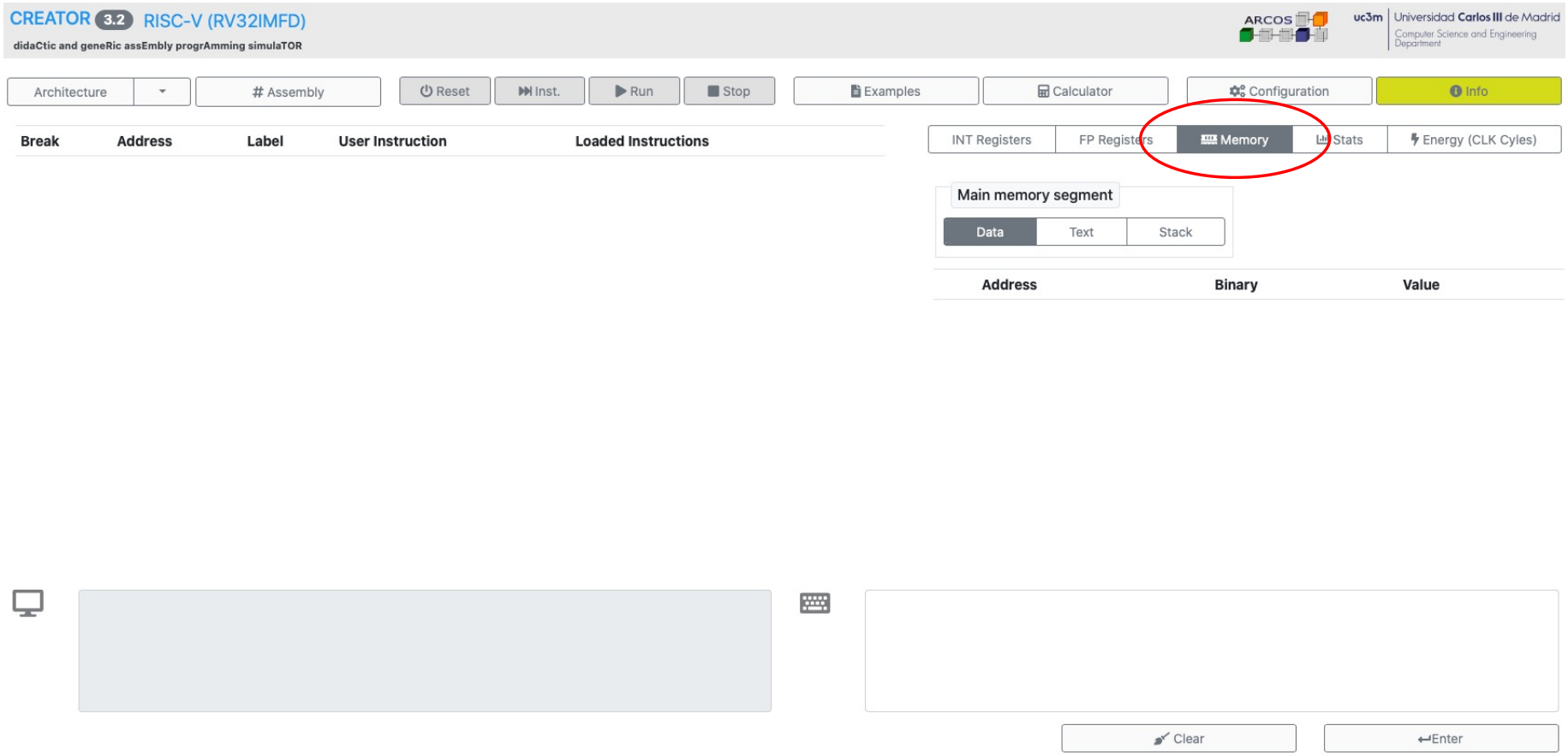
Registros en coma flotante



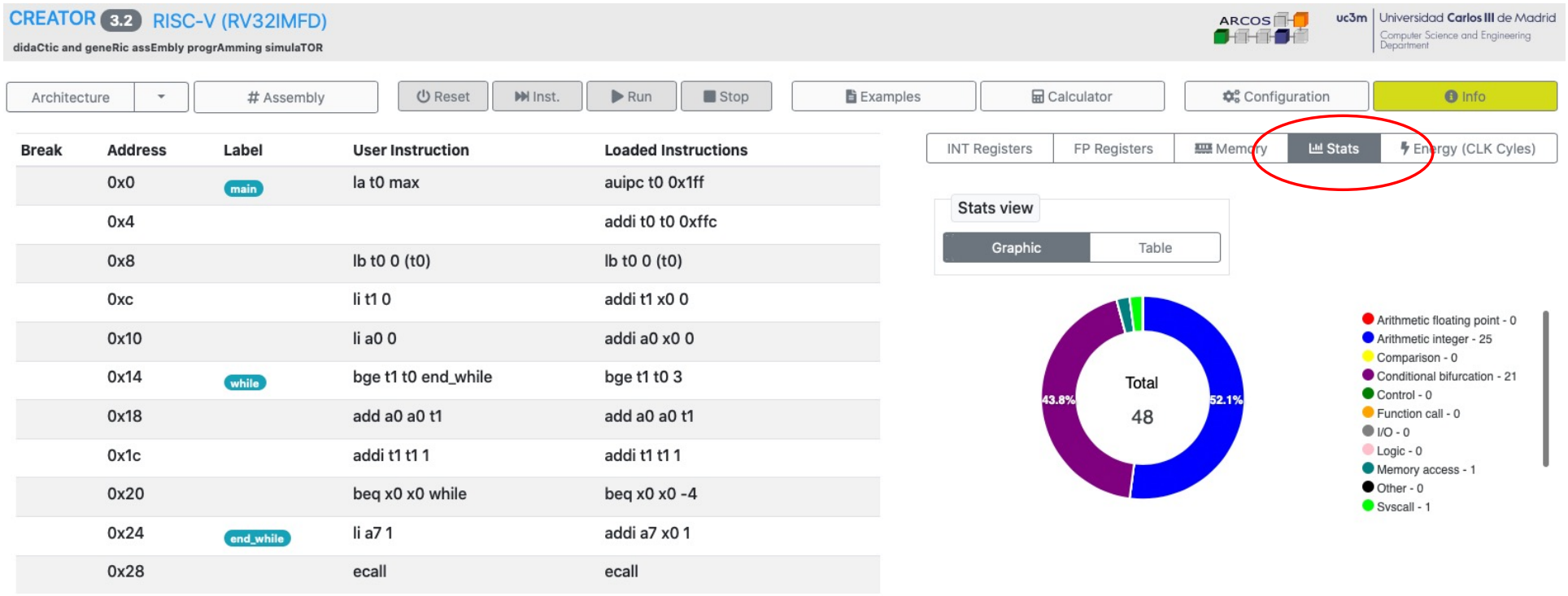
Registros en coma flotante



Contenido de la memoria

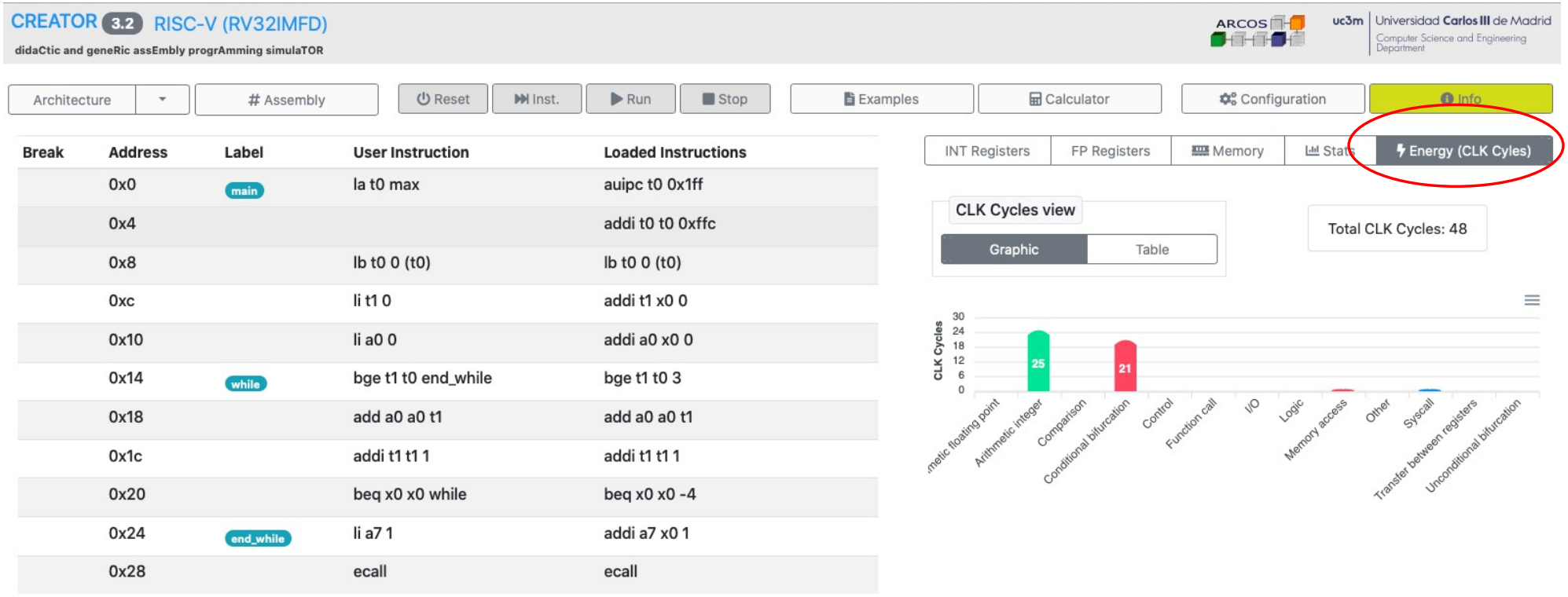


Estadísticas de ejecución

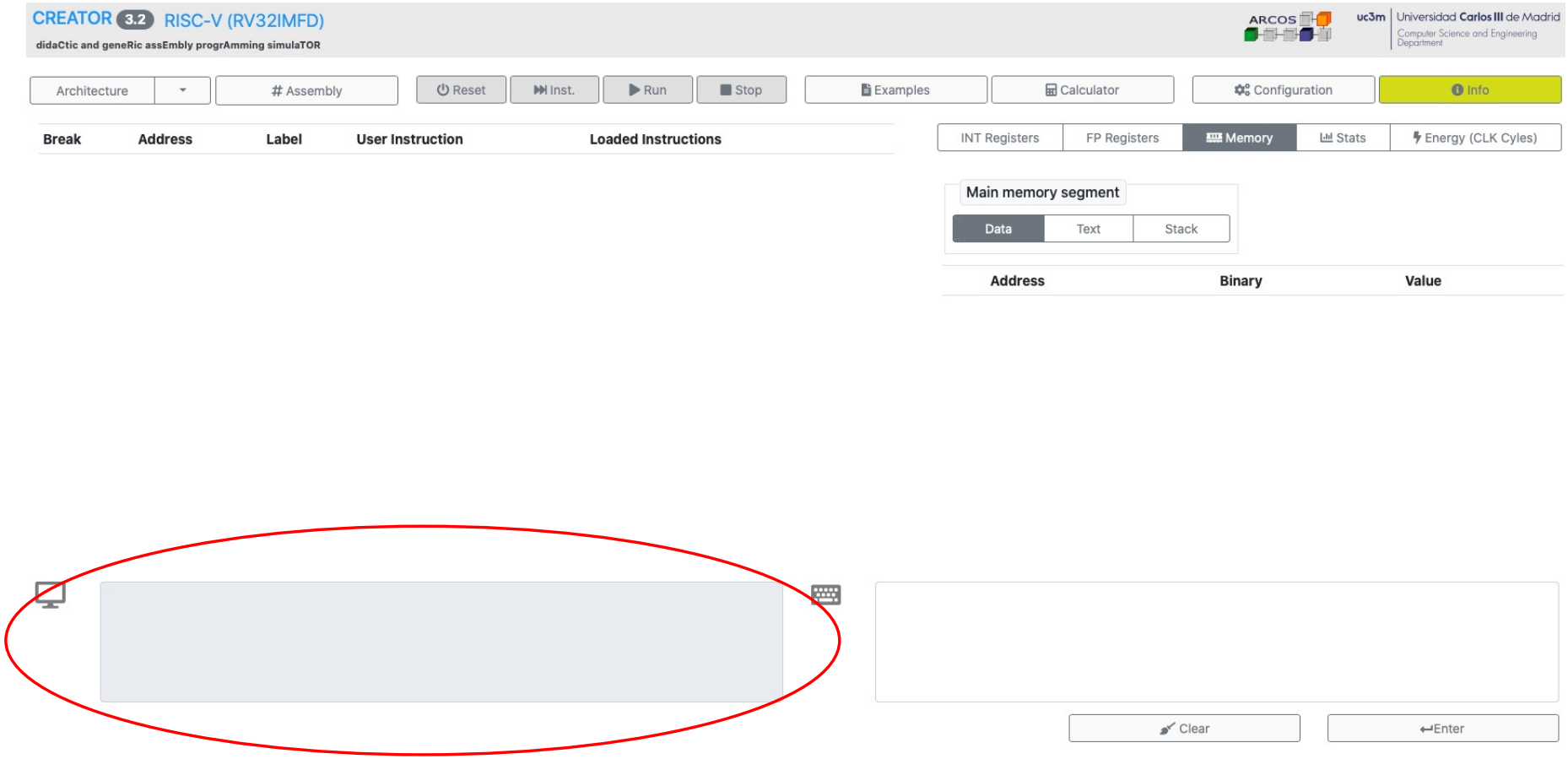


Break	Address	Label	User Instruction	Loaded Instructions
	0x0	main	la t0 max	auipc t0 0x1ff
	0x4			addi t0 t0 0xffc
	0x8		lb t0 0 (t0)	lb t0 0 (t0)
	0xc		li t1 0	addi t1 x0 0
	0x10		li a0 0	addi a0 x0 0
	0x14	while	bge t1 t0 end_while	bge t1 t0 3
	0x18		add a0 a0 t1	add a0 a0 t1
	0x1c		addi t1 t1 1	addi t1 t1 1
	0x20		beq x0 x0 while	beq x0 x0 -4
	0x24	end_while	li a7 1	addi a7 x0 1
	0x28		ecall	ecall

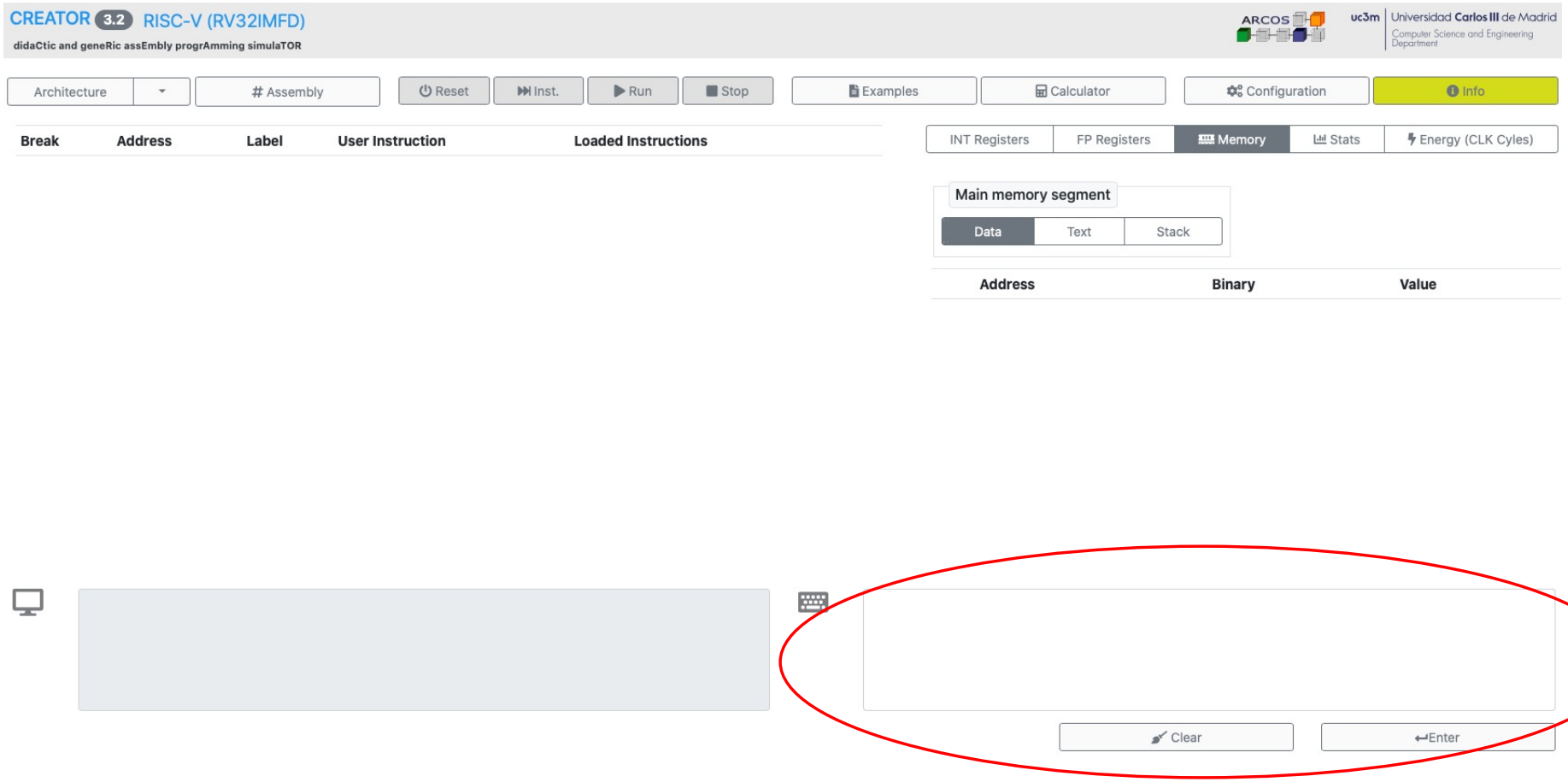
Ciclos ejecutados



Pantalla

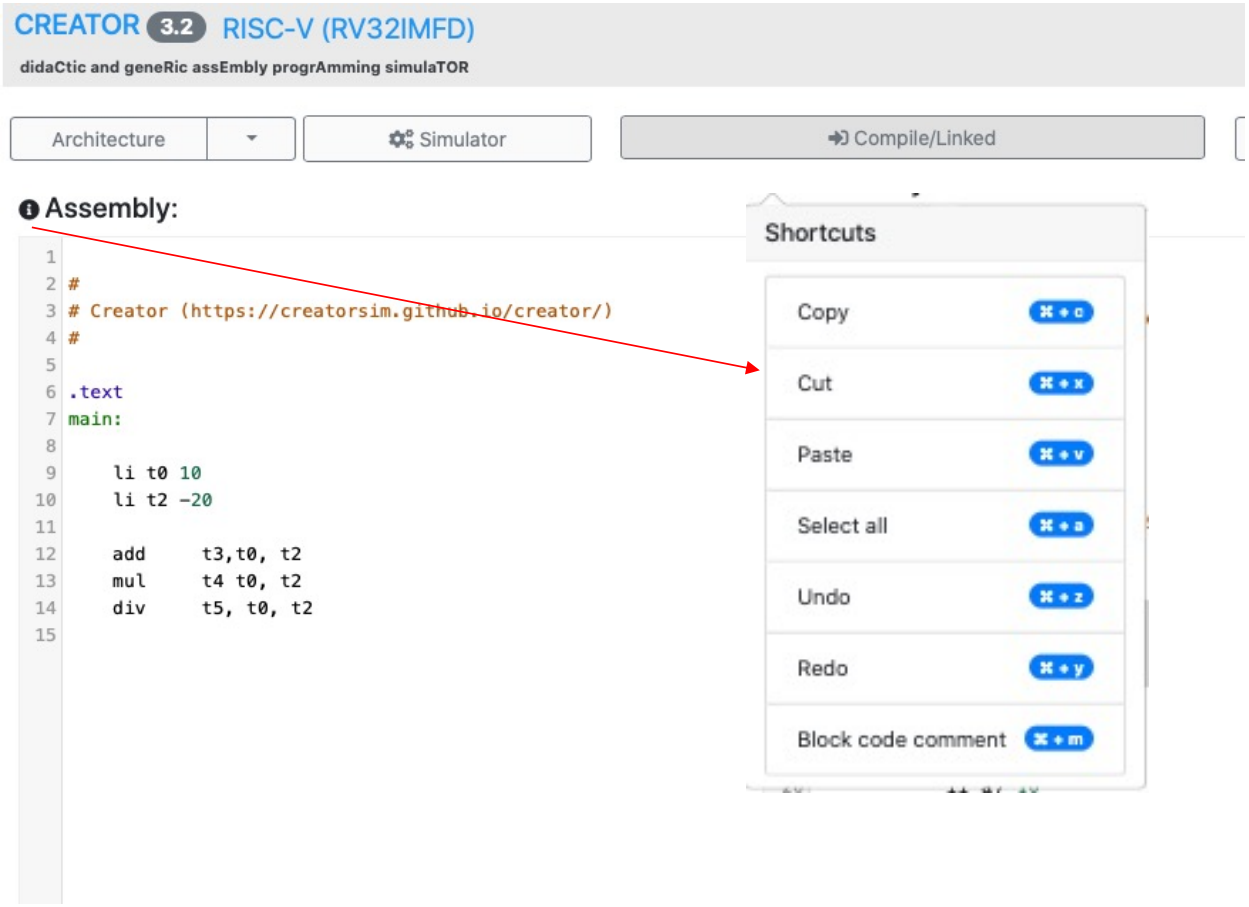

CREATOR 3.2 RISC-V (RV32IMFD)
didaCtic and geneRic assEmbly progrAmming simulaTOR
ARCOS
uc3m Universidad Carlos III de Madrid
Computer Science and Engineering Department
Architecture
Assembly
Reset
Inst.
Run
Stop
Examples
Calculator
Configuration
Info
Break
Address
Label
User Instruction
Loaded Instructions
INT Registers
FP Registers
Memory
Stats
Energy (CLK Cyles)
Main memory segment
Data
Text
Stack
Address
Binary
Value
Clear
Enter

Teclado


CREATOR 3.2 RISC-V (RV32IMFD)
didaCtic and geneRic assEmbly progrAmming simulaTOR
ARCOS
uc3m Universidad Carlos III de Madrid
Computer Science and Engineering Department
Architecture
Assembly
Reset
Inst.
Run
Stop
Examples
Calculator
Configuration
Info
Break
Address
Label
User Instruction
Loaded Instructions
INT Registers
FP Registers
Memory
Stats
Energy (CLK Cyles)
Main memory segment
Data
Text
Stack
Address
Binary
Value
Clear
Enter

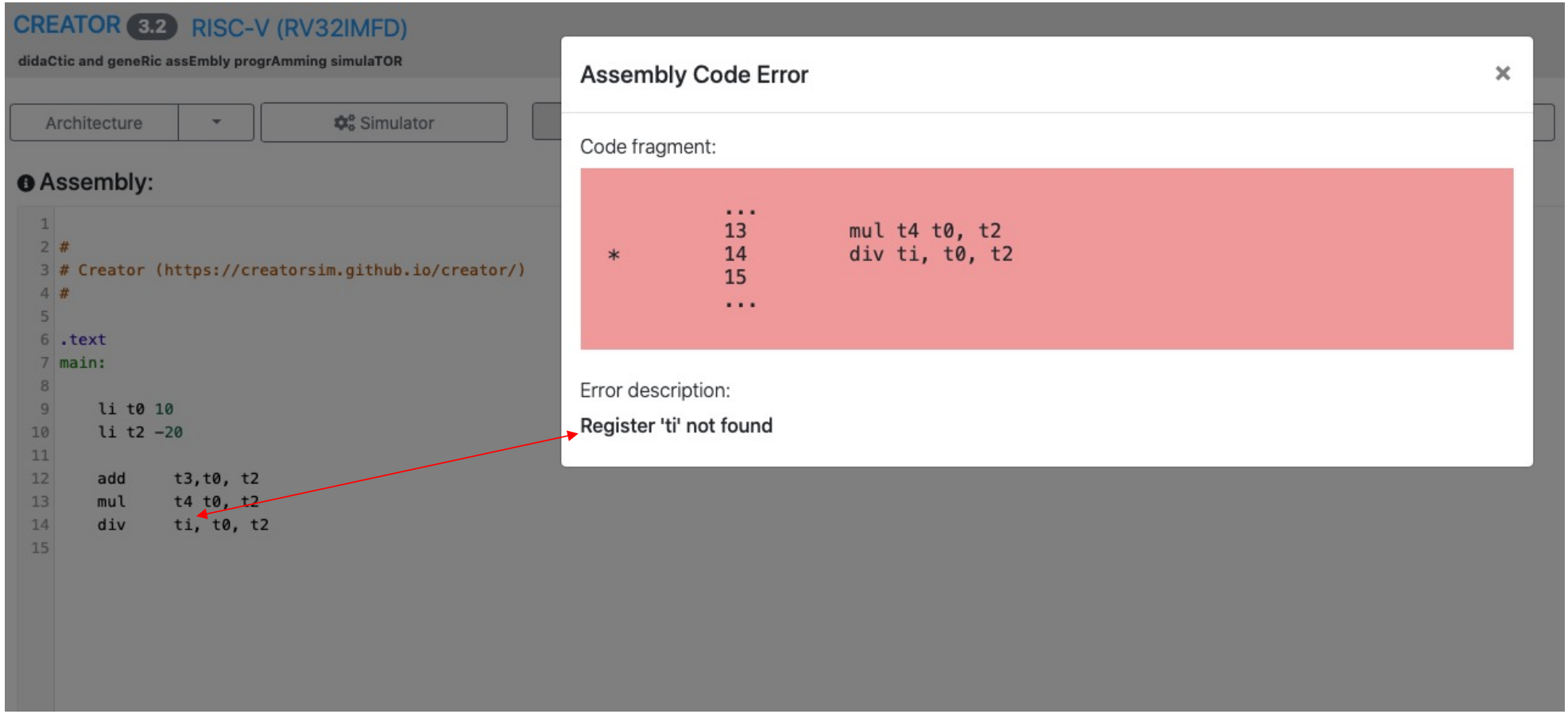
Edición de programas en ensamblador

ejemplo

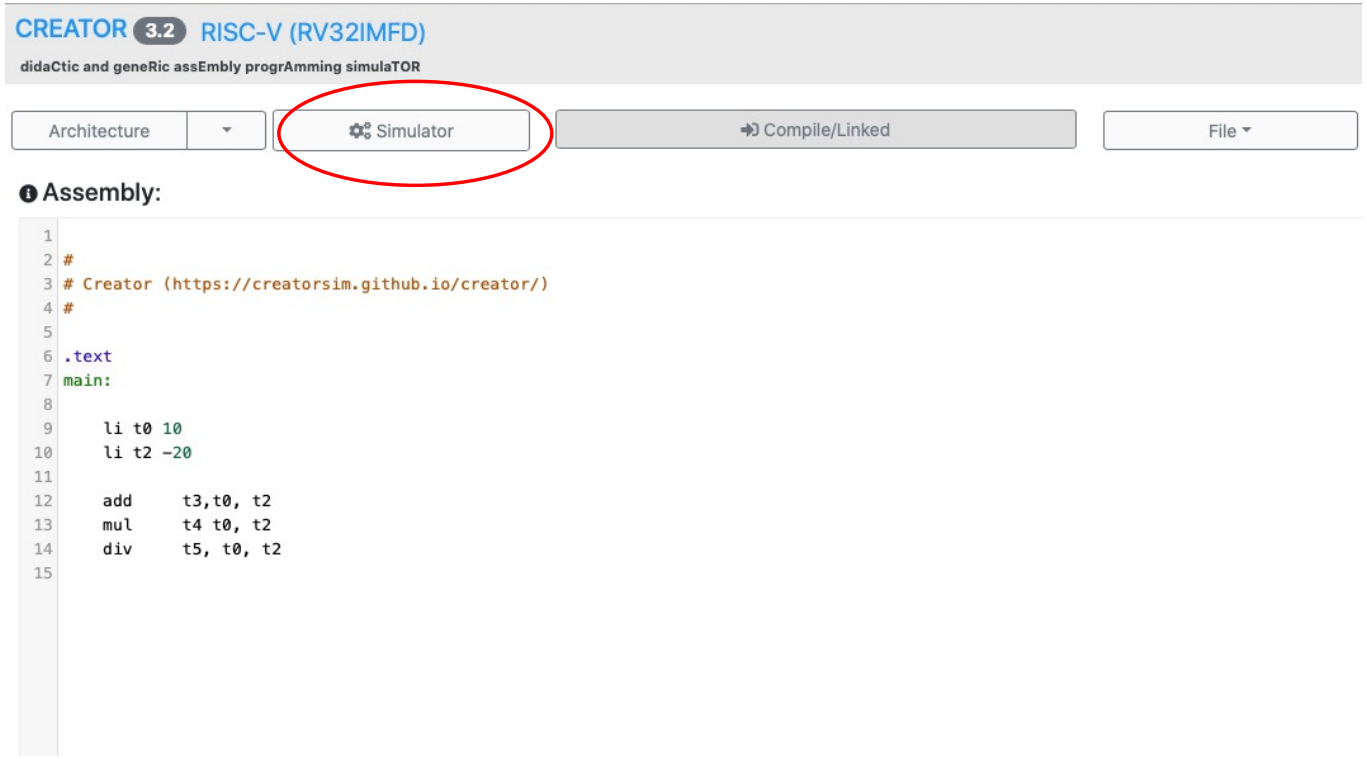


Compilación

Error de compilación

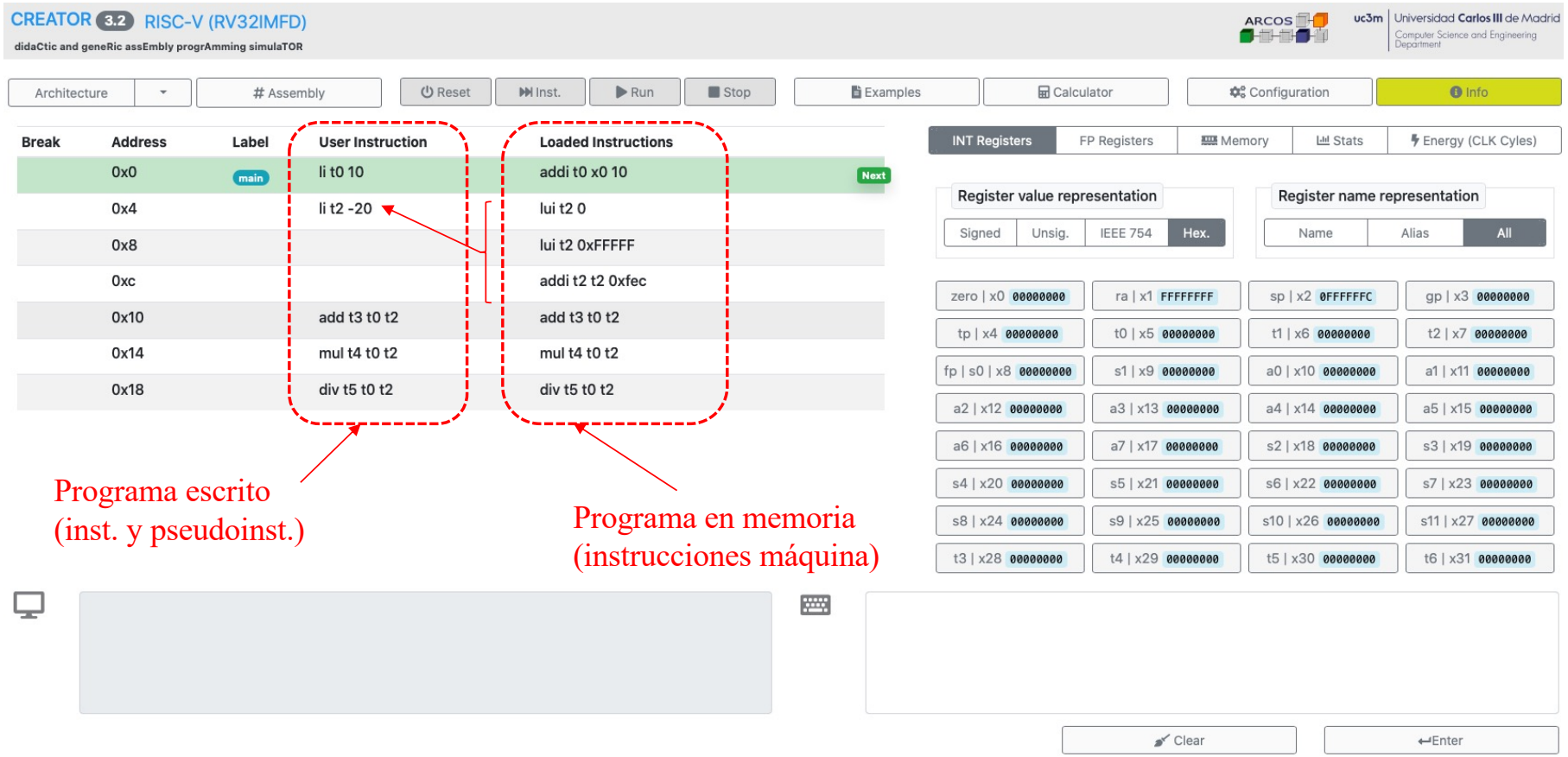


Paso al simulador

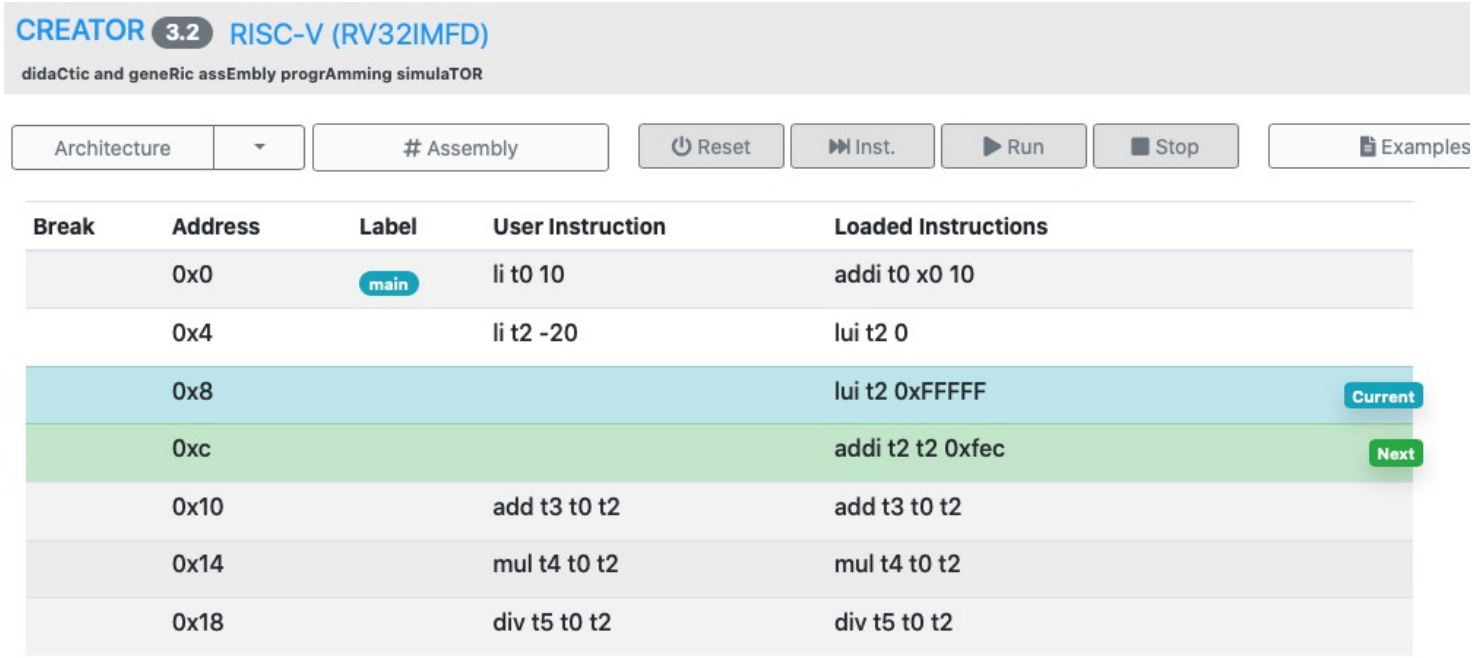


Simulador

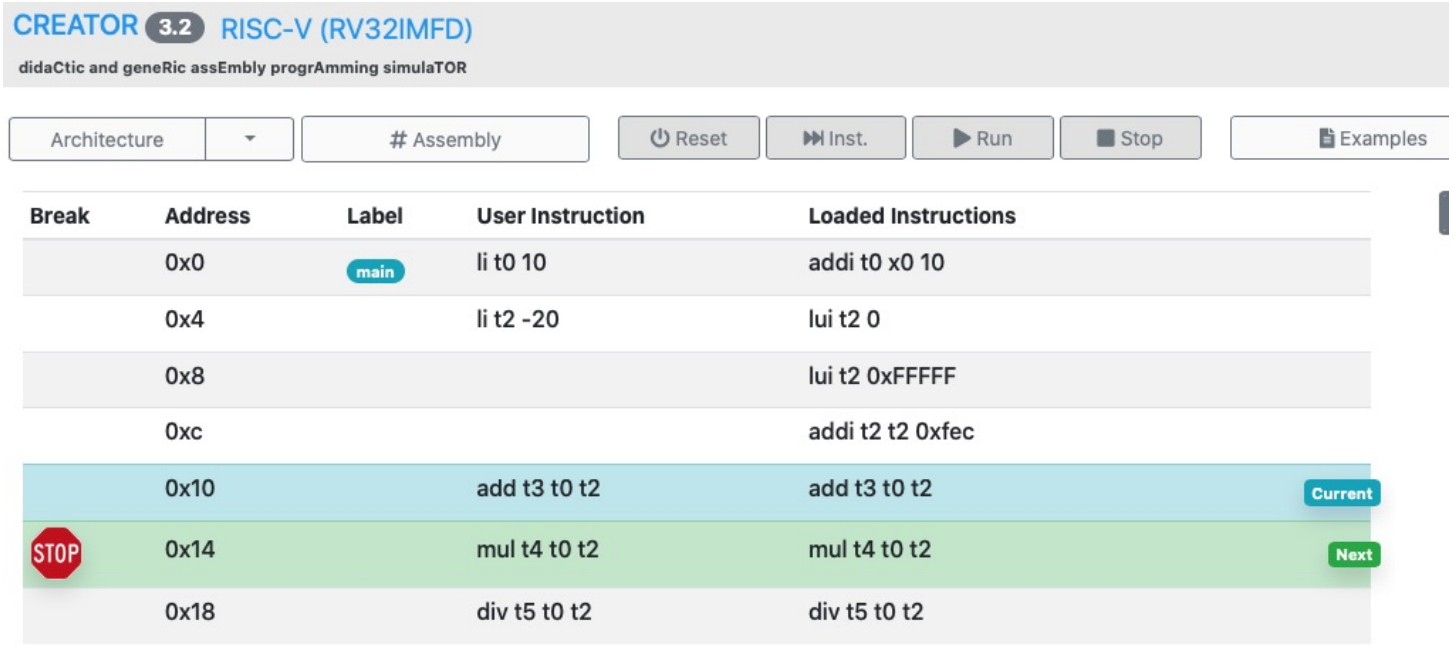
ejemplo



Flujo de ejecución

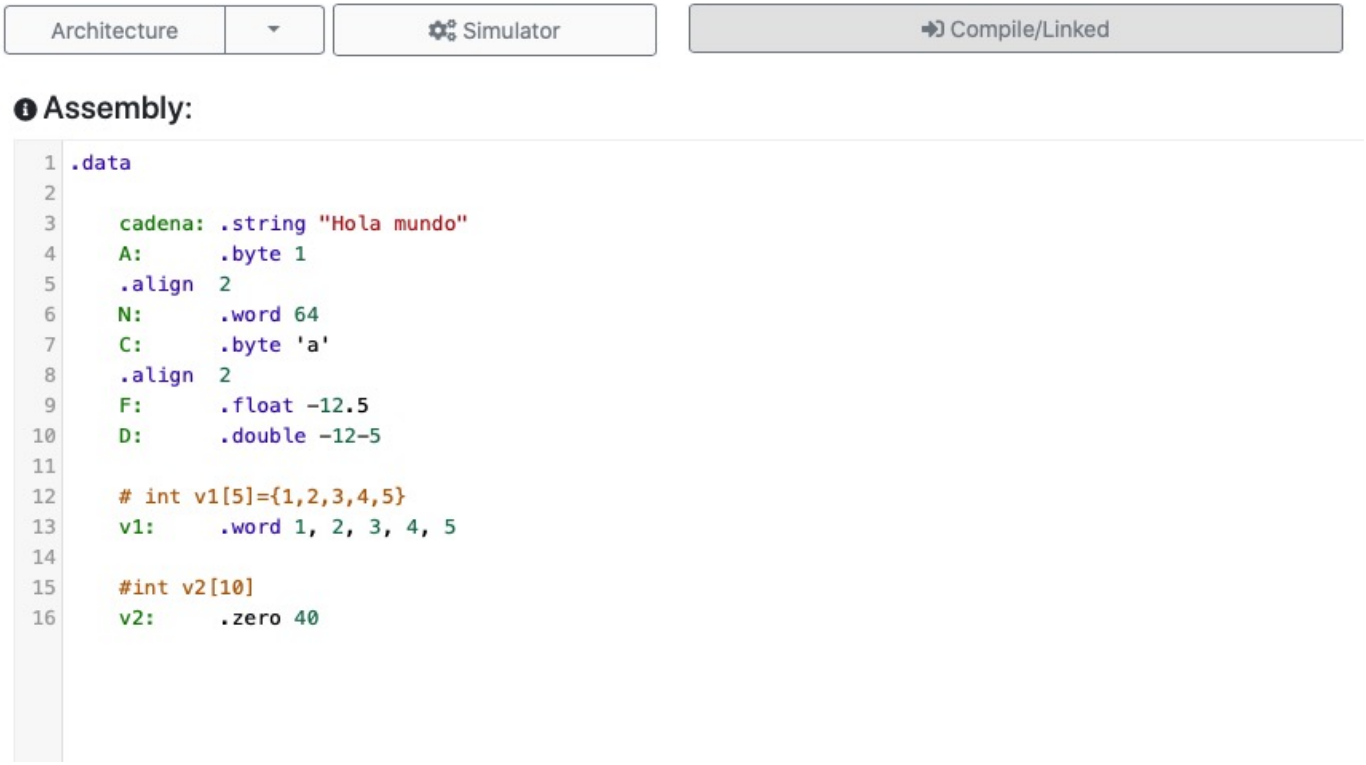


Puntos de ruptura



Segmento de datos

ejemplo



Visualización de datos en memoria

ejemplo

Assembly:

```
1  .data
2
3      cadena: .string "Hola mundo"
4      A:      .byte 1
5      .align  2
6      N:      .word 64
7      C:      .byte 'a'
8      .align  2
9      F:      .float -12.5
10     D:      .double -12-5
11
12     # int v1[5]={1,2,3,4,5}
13     v1:     .word 1, 2, 3, 4, 5
14
15     #int v2[10]
16     v2:     .zero 40
```

Main memory segment

Data | Text | Stack

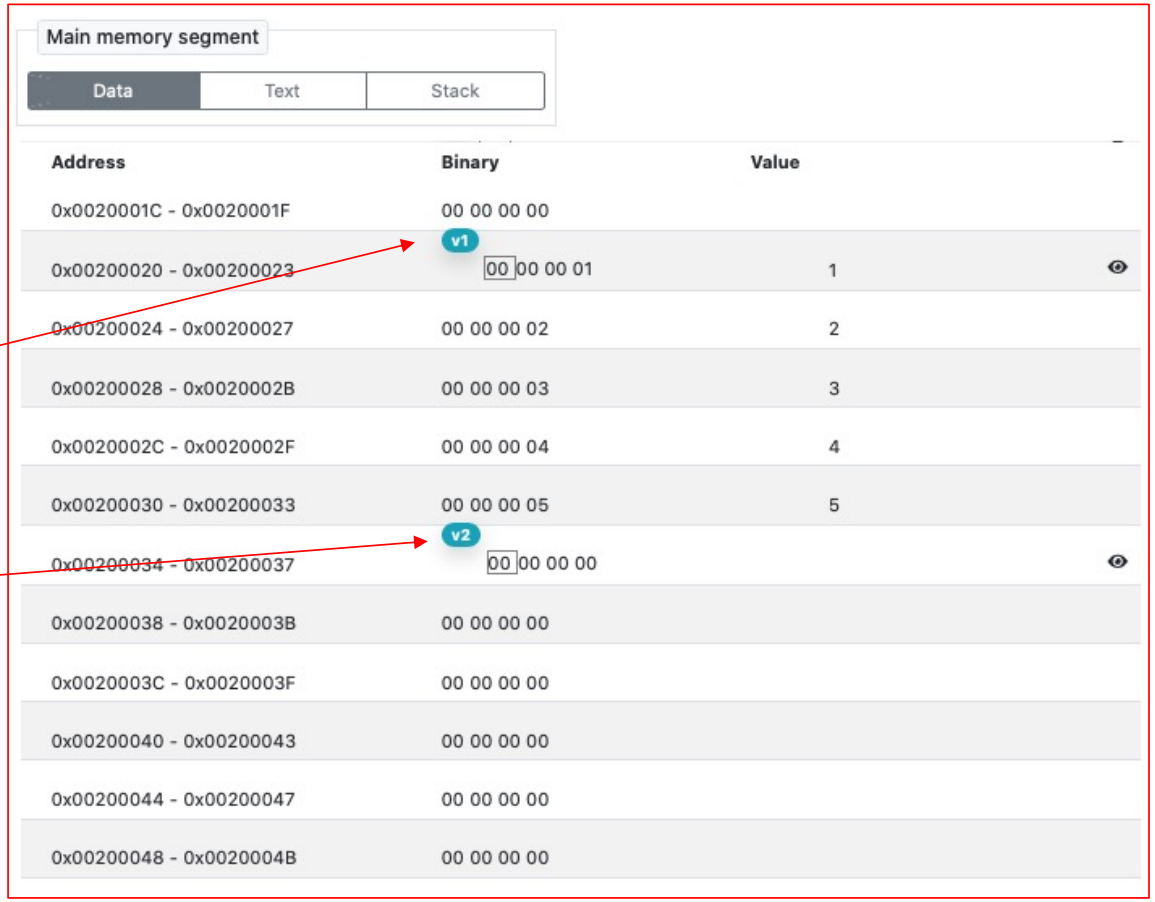
Address	Binary	Value
	cadena	
0x00200000 - 0x00200003	48 6F 6C 61	H, o, l, a
0x00200004 - 0x00200007	20 6D 75 6E	, m, u, n
	A	
0x00200008 - 0x0020000B	64 6F 00 01	d, o, 0, 1
	N	
0x0020000C - 0x0020000F	00 00 00 40	64
	C	
0x00200010 - 0x00200013	61 00 00 00	97
	F	
0x00200014 - 0x00200017	C1 48 00 00	-12.5
	D	
0x00200018 - 0x0020001B	C0 28 00 00	-12-5
0x0020001C - 0x0020001F	00 00 00 00	
	v1	
0x00200020 - 0x00200023	00 00 00 01	1
0x00200024 - 0x00200027	00 00 00 02	2
0x00200028 - 0x0020002B	00 00 00 03	3
0x0020002C - 0x0020002F	00 00 00 04	4

ARCOS

Visualización de datos en memoria

Assembly:

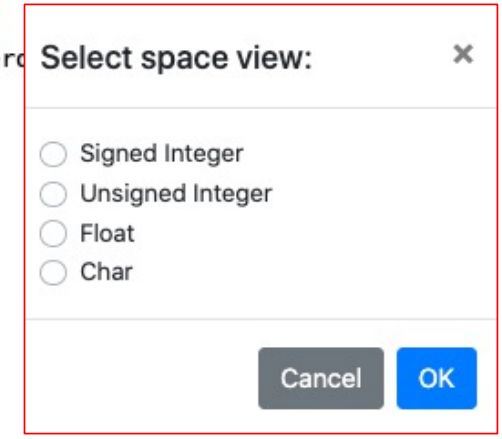
```
1  .data
2
3      cadena: .string "Hola mundo"
4      A:      .byte 1
5      .align  2
6      N:      .word 64
7      C:      .byte 'a'
8      .align  2
9      F:      .float -12.5
10     D:      .double -12-5
11
12     # int v1[5]={1,2,3,4,5}
13     v1:     .word 1, 2, 3, 4, 5
14
15     #int v2[10]
16     v2:     .zero 40
```



Visualización de datos en memoria

Assembly:

```
1  .data
2
3      cadena: .string "Hola mundo"
4      A:      .byte 1
5      .align  2
6      N:      .word 64
7      C:      .byte 'a'
8      .align  2
9      F:      .float -12.5
10     D:      .double -12-5
11
12     # int v1[5]={1,2,3,4,5}
13     v1:     .word 1, 2, 3, 4, 5
14
15     #int v2[10]
16     v2:     .zero
```



Main memory segment

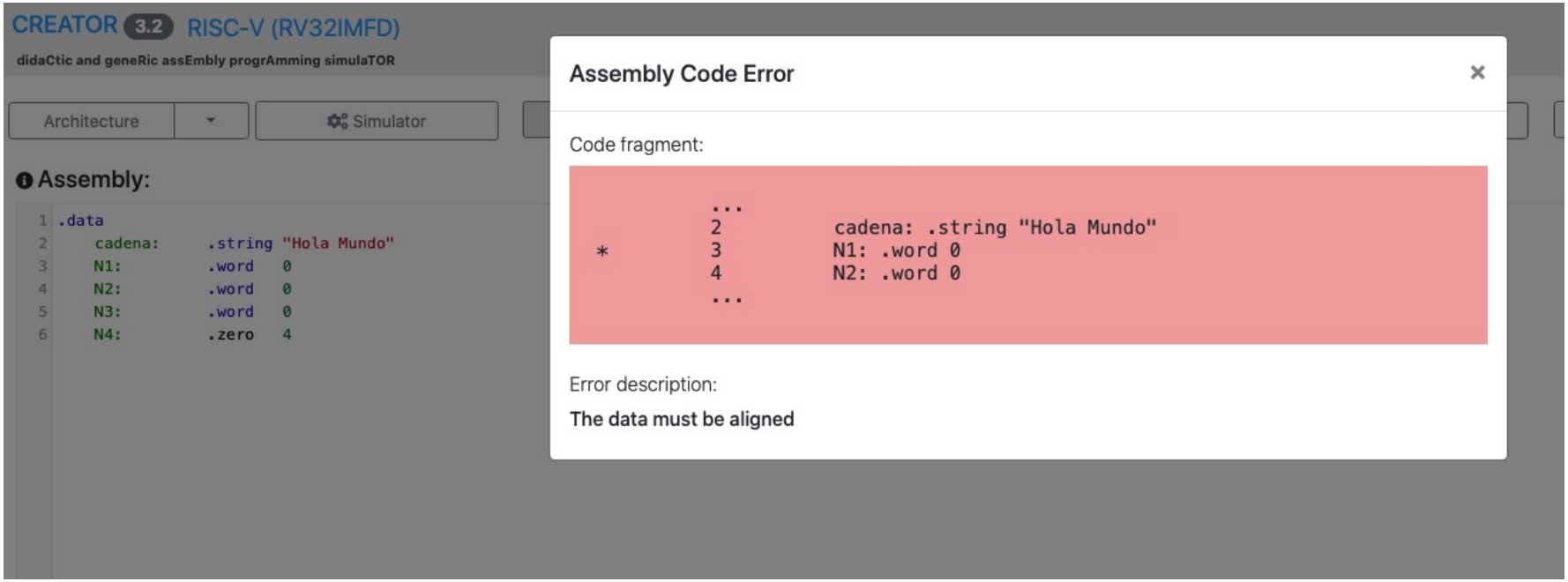
Data | Text | Stack

Address	Binary	Value
0x0020001C - 0x0020001F	00 00 00 00	
0x00200020 - 0x00200023	v1 00 00 00 01	1
0x00200024 - 0x00200027	00 00 00 02	2
0x00200028 - 0x0020002B	00 00 00 03	3
0x0020002C - 0x0020002F	00 00 00 04	4
0x00200030 - 0x00200033	00 00 00 05	5
0x00200034 - 0x00200037	v2 00 00 00 00	
0x00200038 - 0x0020003B	00 00 00 00	
0x0020003C - 0x0020003F	00 00 00 00	
0x00200040 - 0x00200043	00 00 00 00	
0x00200044 - 0x00200047	00 00 00 00	
0x00200048 - 0x0020004B	00 00 00 00	

ARCOS

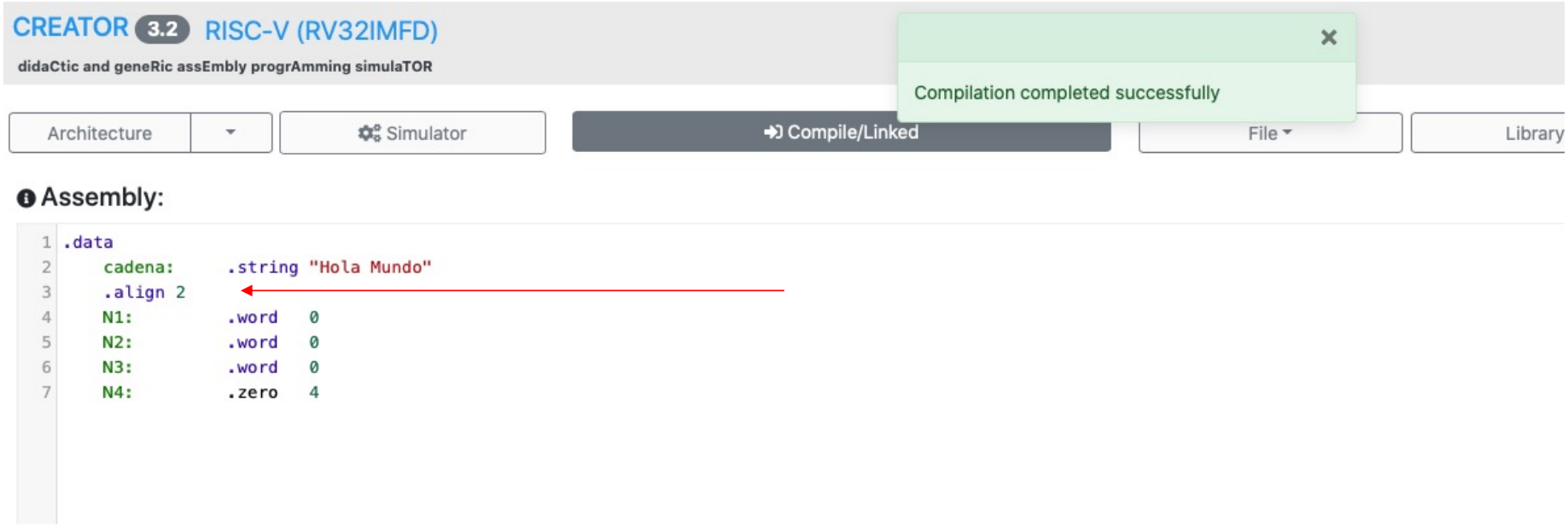
Detección de datos no alineados

ejemplo



Segmento de datos corregido

ejemplo



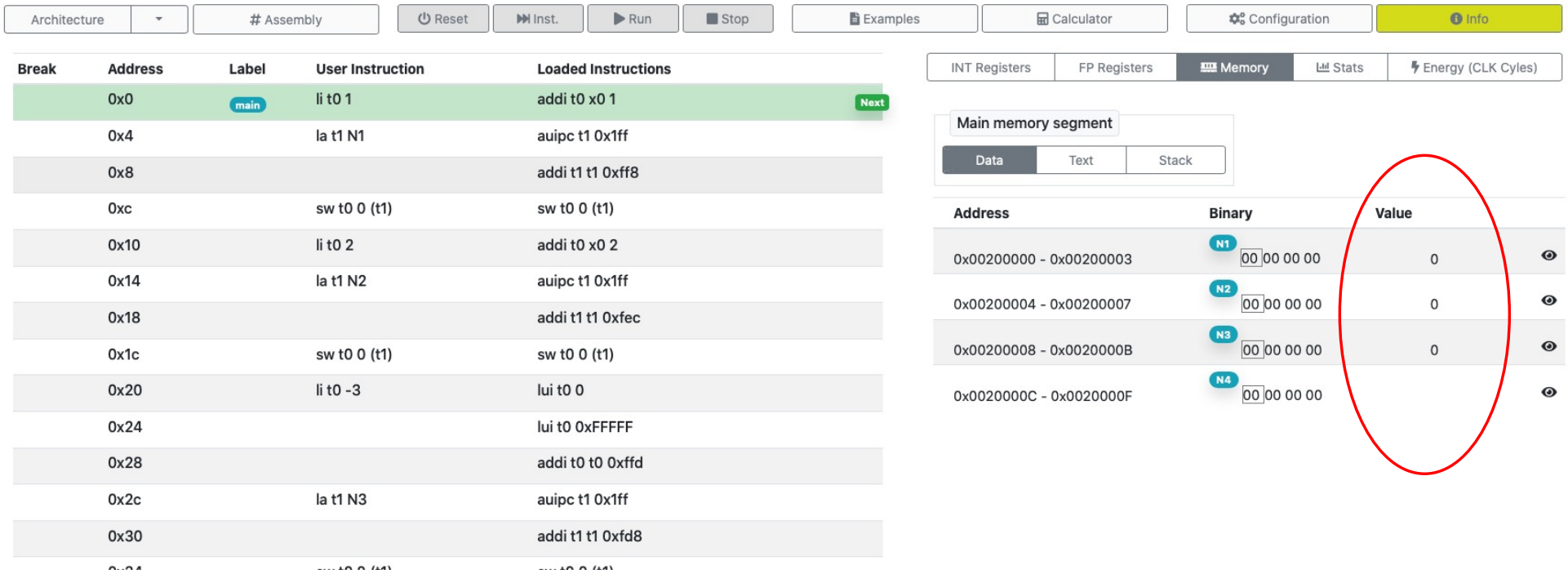
Ejemplo. Ejecutar el siguiente programa

ejemplo

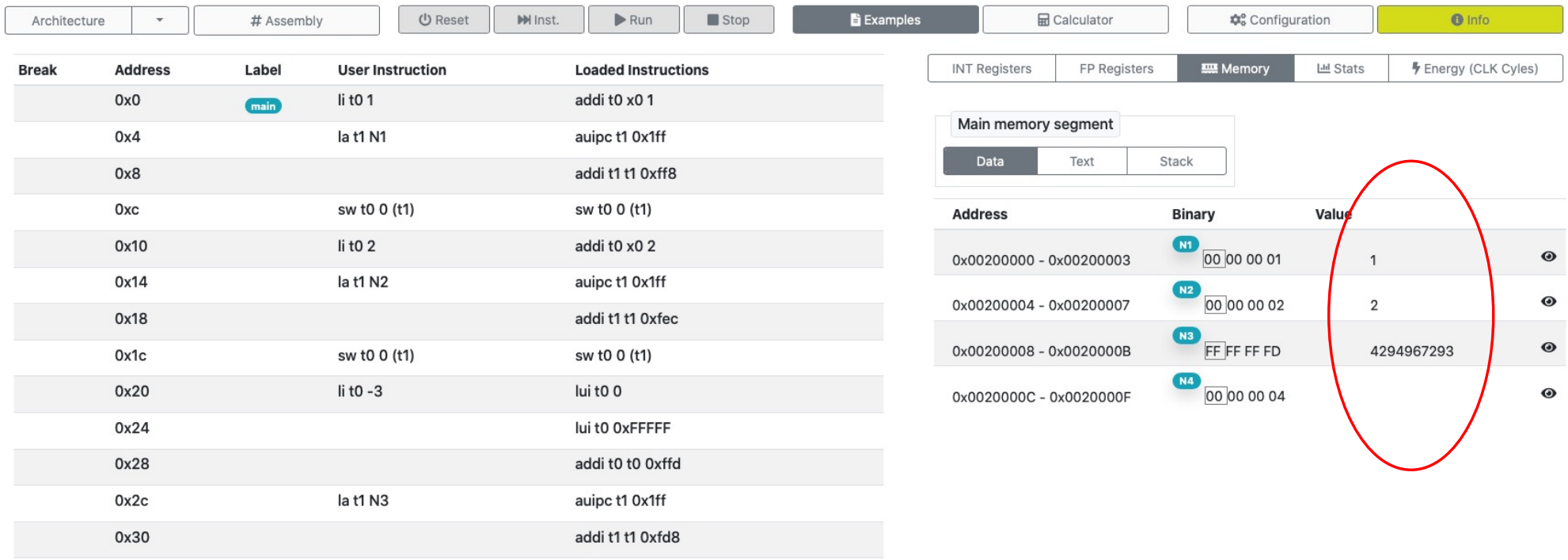
Assembly:

```
.data
    N1: .word   0
    N2: .word   0
    N3: .word   0
    N4: .zero   4

.text
    main:
        li t0, 1
        la t1, N1
        sw t0, 0(t1)

        li t0, 2
        la t1, N2
        sw t0, 0(t1)

        li t0, -3
        la t1, N3
        sw t0, 0(t1)

        li t0, 4
        la t1, N4
        sw t0, 0(t1)
```

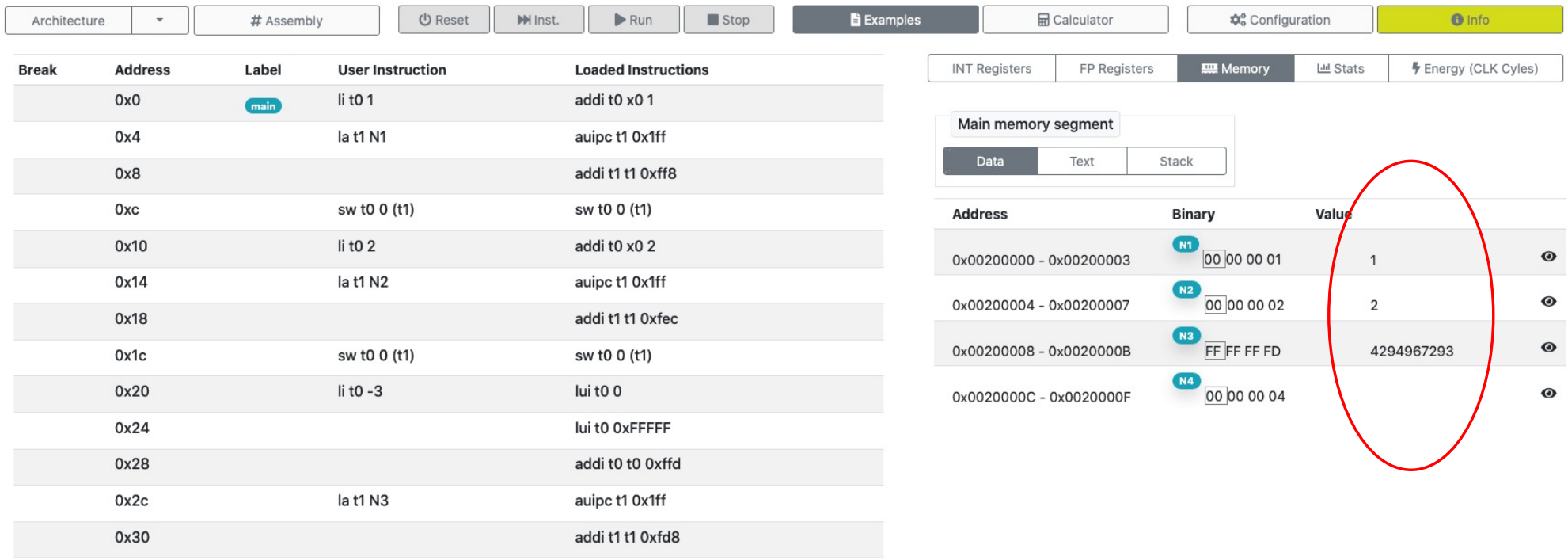
Ejemplo: antes de la ejecución



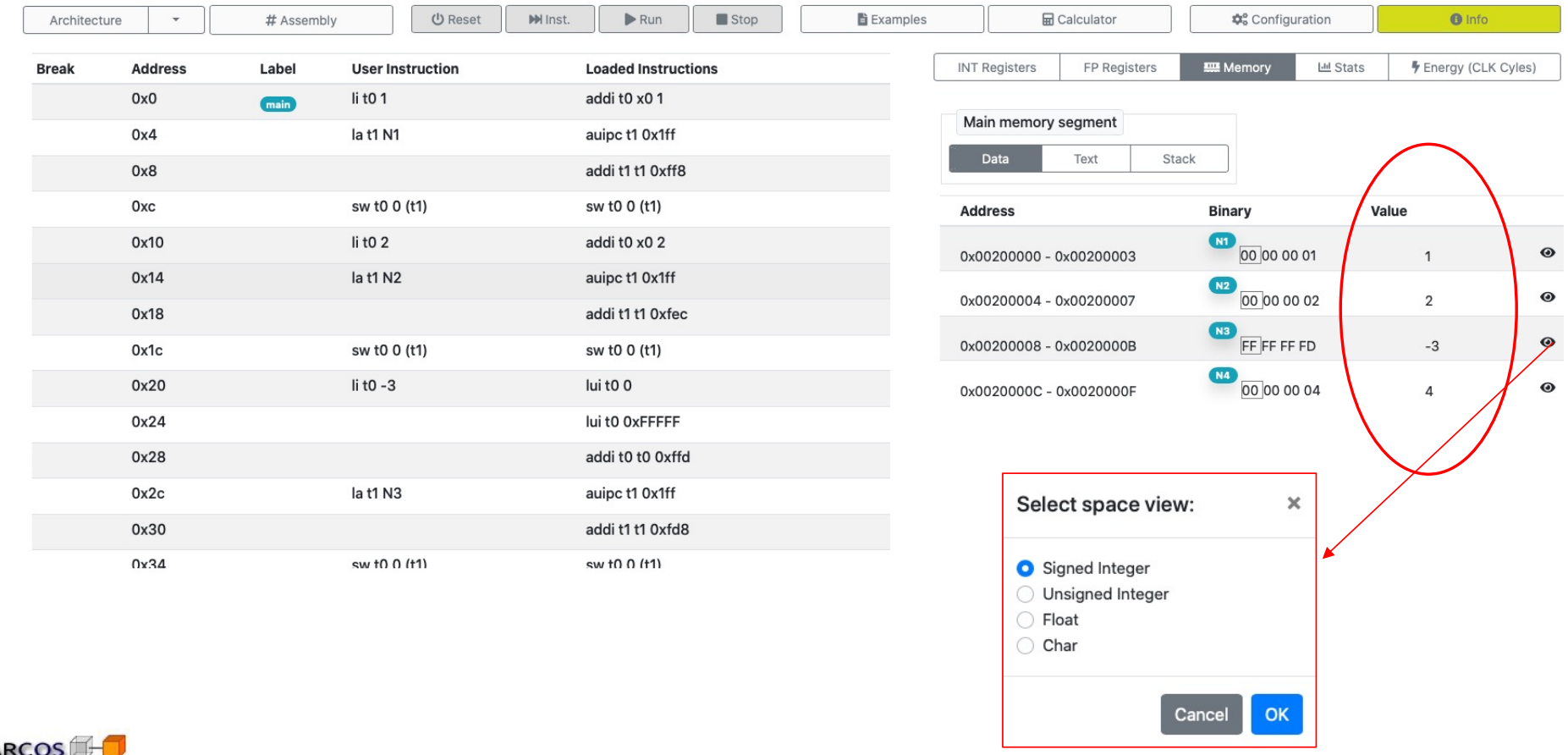
Ejemplo: después de la ejecución



Ejemplo: después de la ejecución



Ejemplo: después de la ejecución



Visualización del segmento de texto
Instrucciones y pseudoinstrucciones

ejemplo

Break	Address	Label	User Instruction	Loaded Instructions
	0x0	main	li t0 1	addi t0 x0 1 Next
	0x4		la t1 N1	auipc t1 0x1ff
	0x8			addi t1 t1 0xff8
	0xc		sw t0 0 (t1)	sw t0 0 (t1)
	0x10		li t0 2	addi t0 x0 2
	0x14		la t1 N2	auipc t1 0x1ff
	0x18			addi t1 t1 0xfec
	0x1c		sw t0 0 (t1)	sw t0 0 (t1)
	0x20		li t0 -3	lui t0 0
	0x24			lui t0 0xFFFFF
	0x28			addi t0 t0 0xffd
	0x2c		la t1 N3	auipc t1 0x1ff
	0x30			addi t1 t1 0xfd8
	0x34		sw t0 0 (t1)	sw t0 0 (t1)

INT Registers | FP Registers | Memory | Stats | Energy (CLK Cyles)

Main memory segment

Data | Text | Stack

	Address	Binary	Value
PC gp fp	0x00000000 - 0x00000003	main 00 10 02 93	addi t0 x0 1
	0x00000004 - 0x00000007	00 1F F3 17	auipc t1 0x1ff
	0x00000008 - 0x0000000B	FF 83 03 13	addi t1 t1 0xff8
	0x0000000C - 0x0000000F	00 53 20 23	sw t0 0 (t1)
	0x00000010 - 0x00000013	00 20 02 93	addi t0 x0 2
	0x00000014 - 0x00000017	00 1F F3 17	auipc t1 0x1ff
	0x00000018 - 0x0000001B	FE C3 03 13	addi t1 t1 0xfec
	0x0000001C - 0x0000001F	00 53 20 23	sw t0 0 (t1)

Programa escrito
(inst. y pseudoinst.)

Programa en memoria
(instrucciones máquina)

Visualización del segmento de texto
Flujo de ejecución

ejemplo

Break	Address	Label	User Instruction	Loaded Instructions	
	0x0	main	li t0 1	addi t0 x0 1	
	0x4		la t1 N1	auipc t1 0x1ff	
	0x8			addi t1 t1 0xff8	
	0xc		sw t0 0 (t1)	sw t0 0 (t1)	
	0x10		li t0 2	addi t0 x0 2	
	0x14		la t1 N2	auipc t1 0x1ff	Current
	0x18			addi t1 t1 0xfec	Next
	0x1c		sw t0 0 (t1)	sw t0 0 (t1)	
	0x20		li t0 -3	lui t0 0	
	0x24			lui t0 0xFFFFF	
	0x28			addi t0 t0 0xffd	
	0x2c		la t1 N3	auipc t1 0x1ff	
	0x30			addi t1 t1 0xfd8	
	0x34		sw t0 0 (t1)	sw t0 0 (t1)	

INT Registers | FP Registers | Memory | Stats | Energy (CLK Cyles)

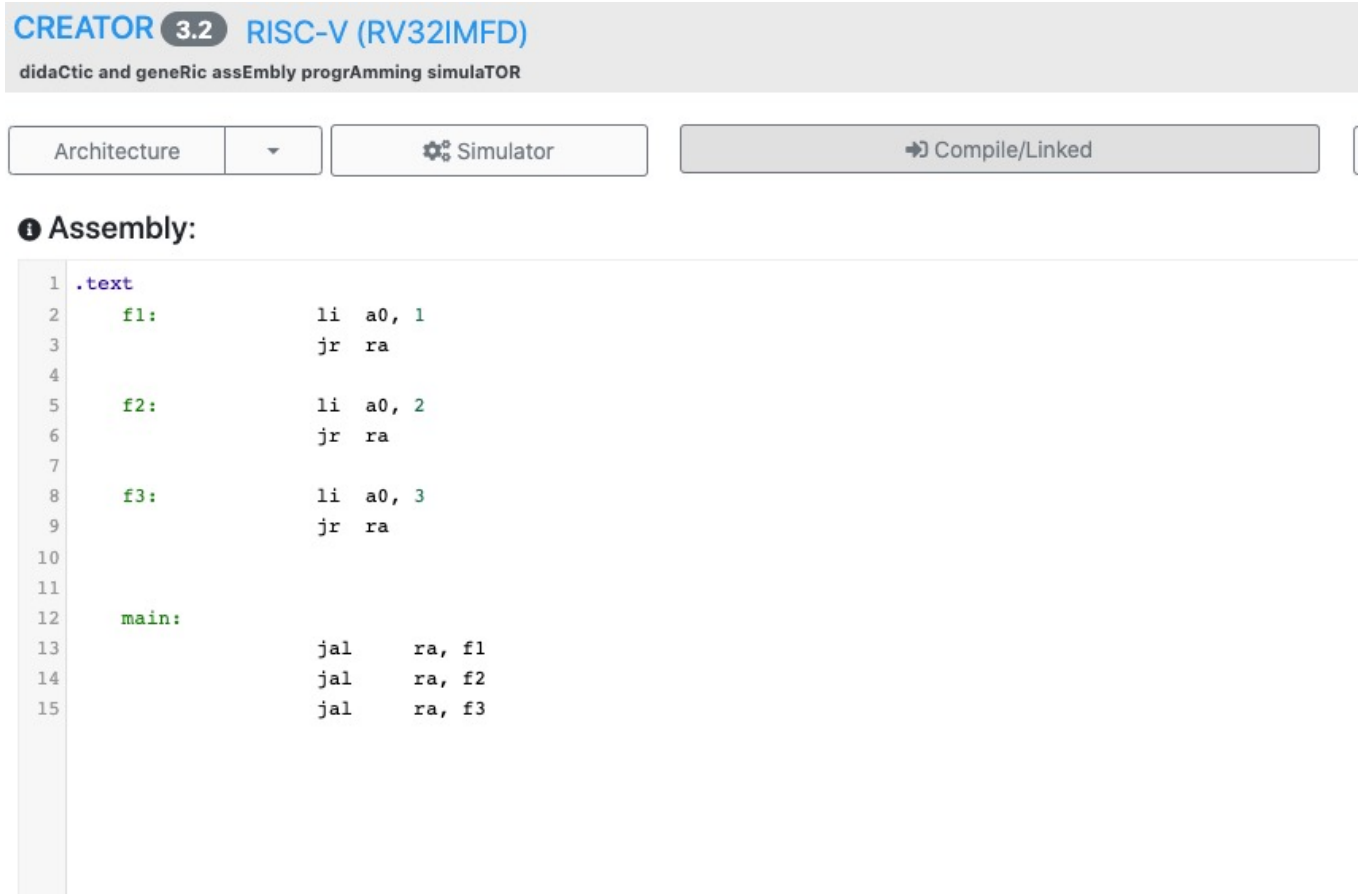
Main memory segment

Data | Text | Stack

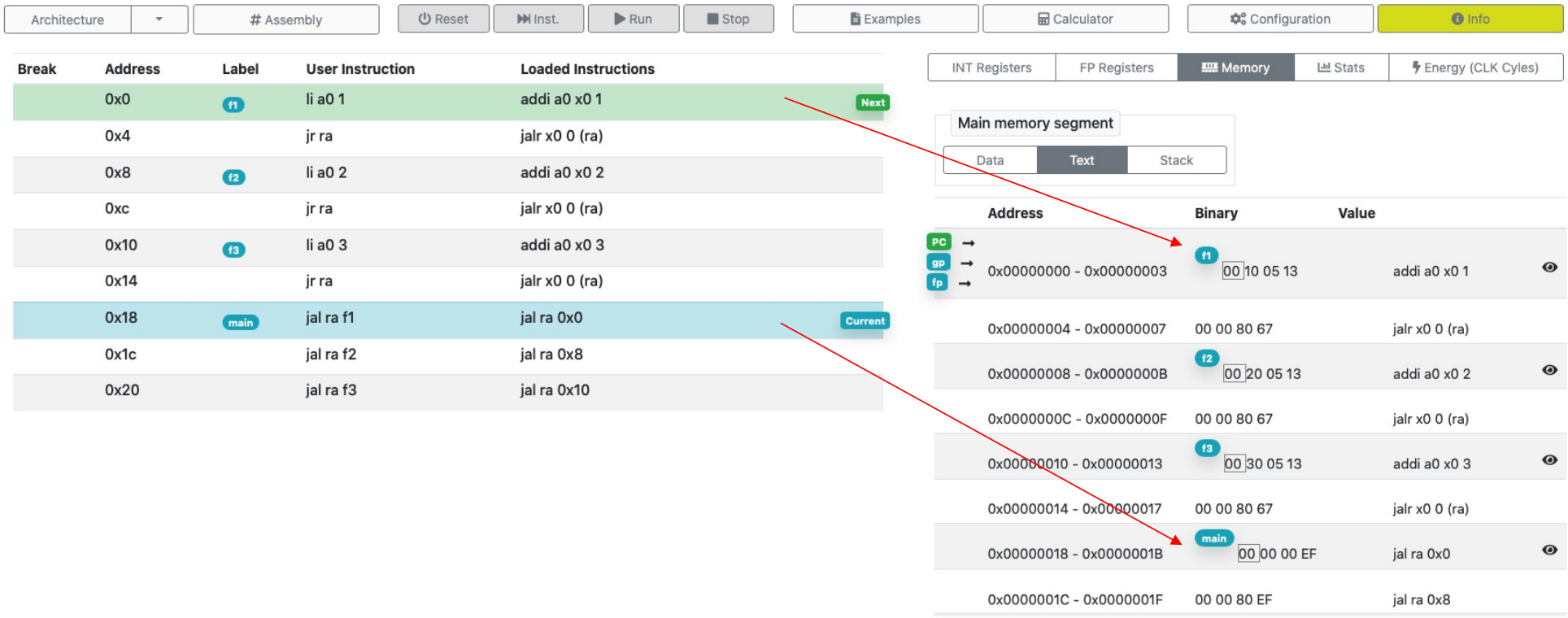
	Address	Binary	Value
	0x00000008 - 0x0000000B	FF 83 03 13	addi t1 t1 0xff8
	0x0000000C - 0x0000000F	00 53 20 23	sw t0 0 (t1)
	0x00000010 - 0x00000013	00 20 02 93	addi t0 x0 2
	0x00000014 - 0x00000017	00 1F F3 17	auipc t1 0x1ff
PC →	0x00000018 - 0x0000001B	FE C3 03 13	addi t1 t1 0xfec
	0x0000001C - 0x0000001F	00 53 20 23	sw t0 0 (t1)
	0x00000020 - 0x00000023	00 00 02 B7	lui t0 0
	0x00000024 - 0x00000027	FF FF F2 B7	lui t0 0xFFFFF
	0x00000028 - 0x0000002B	FF D2 82 93	addi t0 t0 0xffd

Visualización del segmento de texto
Varias funciones

ejemplo



Visualización del segmento de texto
Varias funciones



Llamadas al sistema

ejemplo

CREATOR 3.2 RISC-V (RV32IMFD)

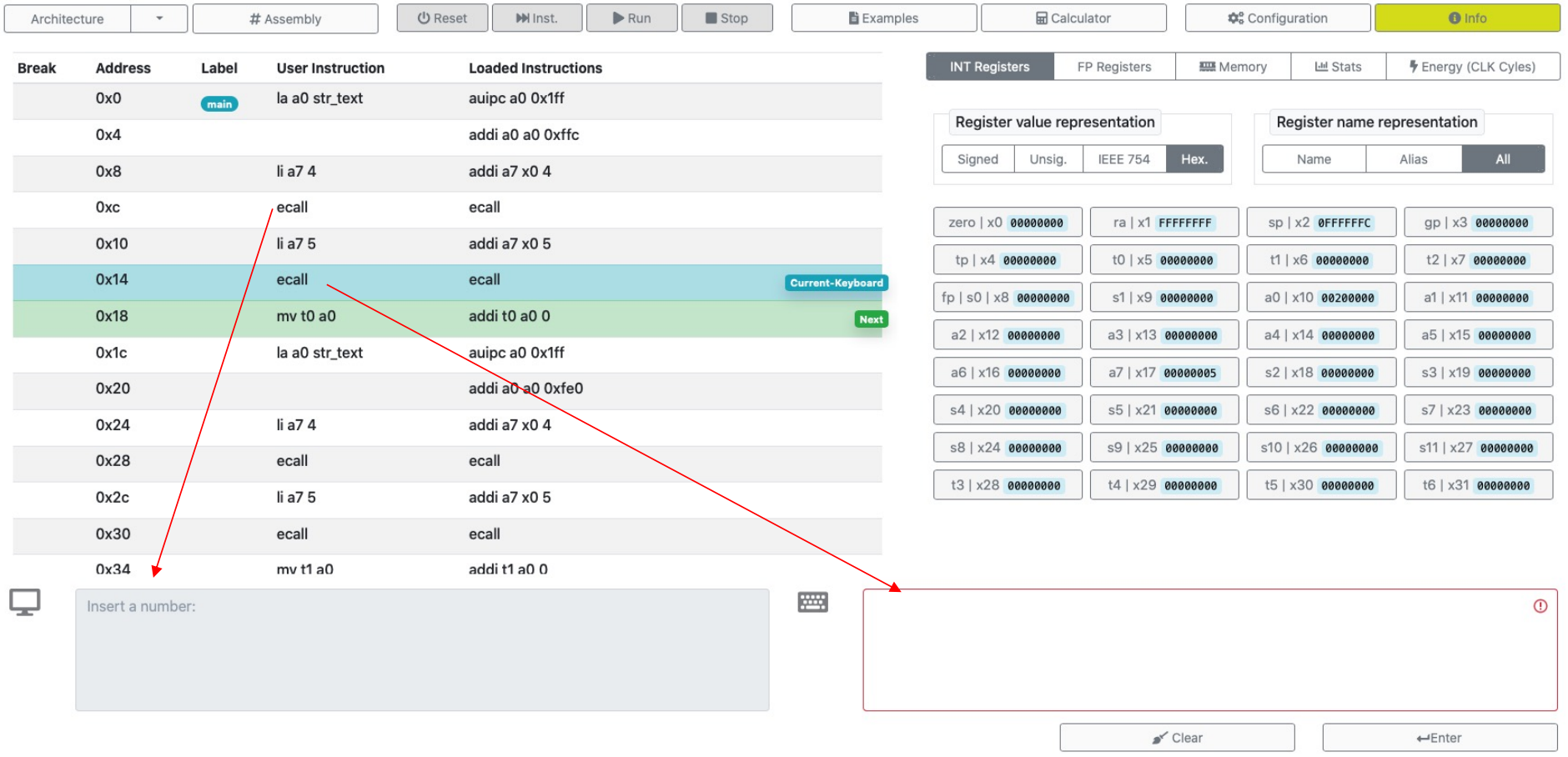
didaCtic and geneRic assEmbly progrAmming simulaTOR

Architecture | Simulator | Compile/Linked | File

Assembly:

```
.data
    str_text: .string "Insert a number: "
    str_result: .string "Result = "



.text
main:
    # print "Insert a number: "
    la a0 str_text
    li a7 4
    ecall

    # read int
    li a7 5
    ecall

    mv t0, a0

    # print "Insert a number: "
    la a0 str_text
    li a7 4
    ecall

      # read int
    li a7 5
    ecall
    mv t1, a0

    # print "Result = "
    la a0 str_result
    li a7 4
```

Llamadas al sistema



Ejemplo: ejecutar el siguiente programa

ejemplo

Assembly:

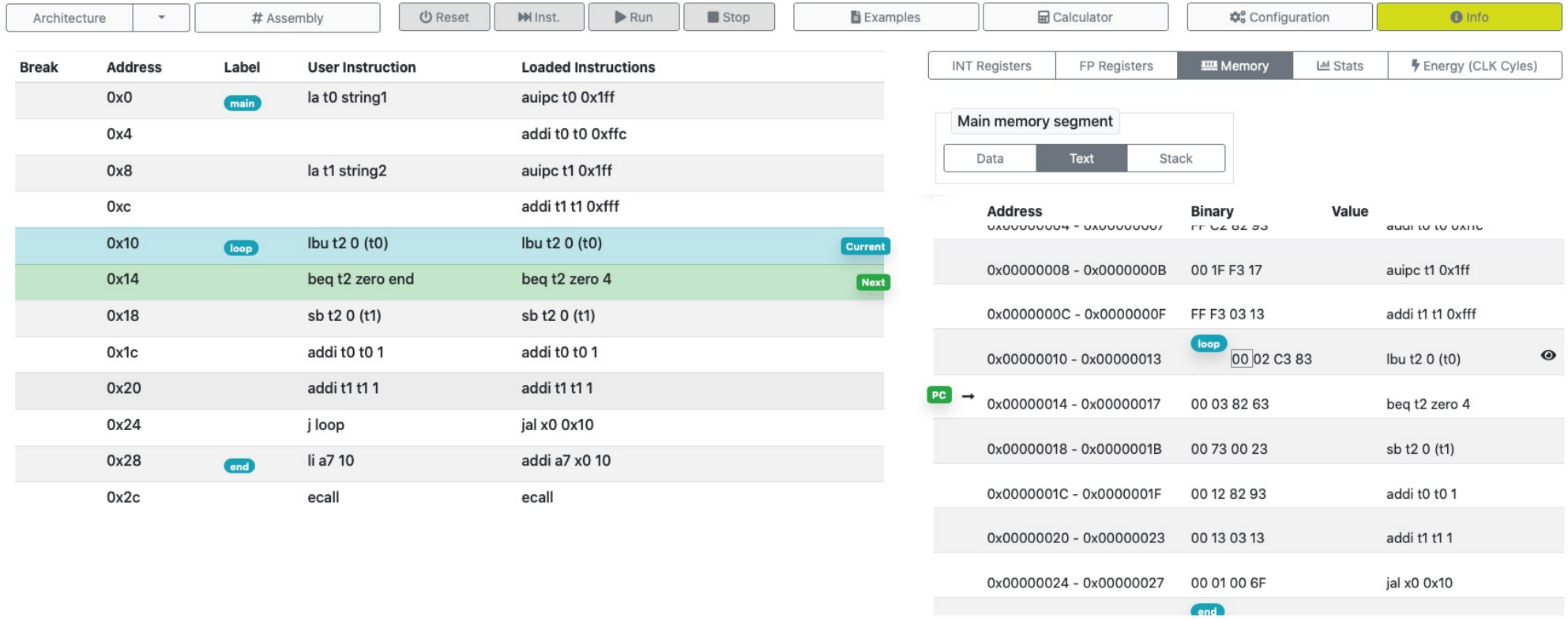
```
1  #
2  # Creator (https://creatorsim.github.io/creator/)
3  #
4
5   .data
6
7      string1:     .string  "Hola Mundo"
8      string2:     .zero 32
9
10 .text
11     main:
12                      la   t0, string1
13                      la   t1, string2
14
15                loop: lbu      t2, 0(t0)
16                      beq      t2, zero, end
17                      sb       t2, 0(t1)
18                      addi     t0, t0, 1
19                      addi     t1, t1, 1
20                      j        loop
21                end:
22                      li a7, 10
23                      ecall
24
```

Main memory segment

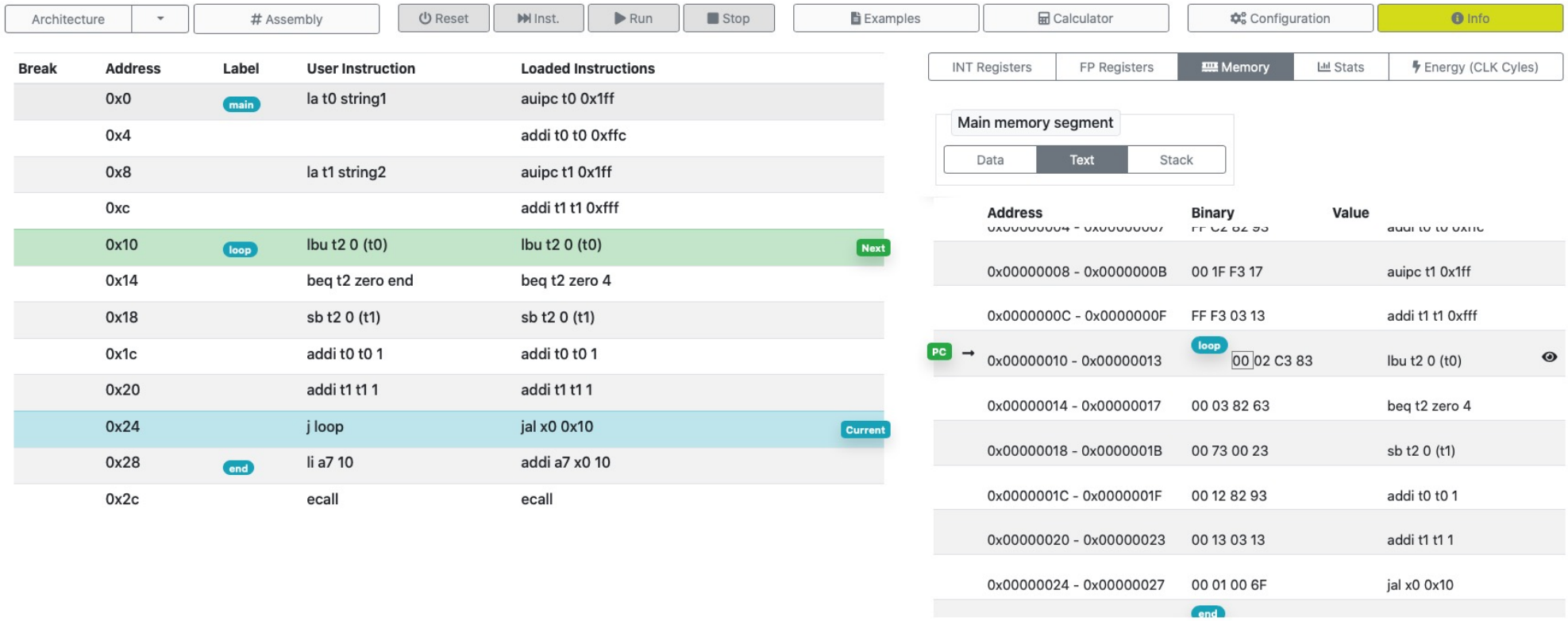
Data | Text | Stack

Address	Binary	Value
0x00200000 - 0x00200003	string1 48 6F 6C 61	H, o, l, a
0x00200004 - 0x00200007	20 4D 75 6E	, M, u, n
0x00200008 - 0x0020000B	64 6F 00 string2 00	d, o, 0
0x0020000C - 0x0020000F	00 00 00 00	
0x00200010 - 0x00200013	00 00 00 00	
0x00200014 - 0x00200017	00 00 00 00	
0x00200018 - 0x0020001B	00 00 00 00	
0x0020001C - 0x0020001F	00 00 00 00	

Ejecución del programa



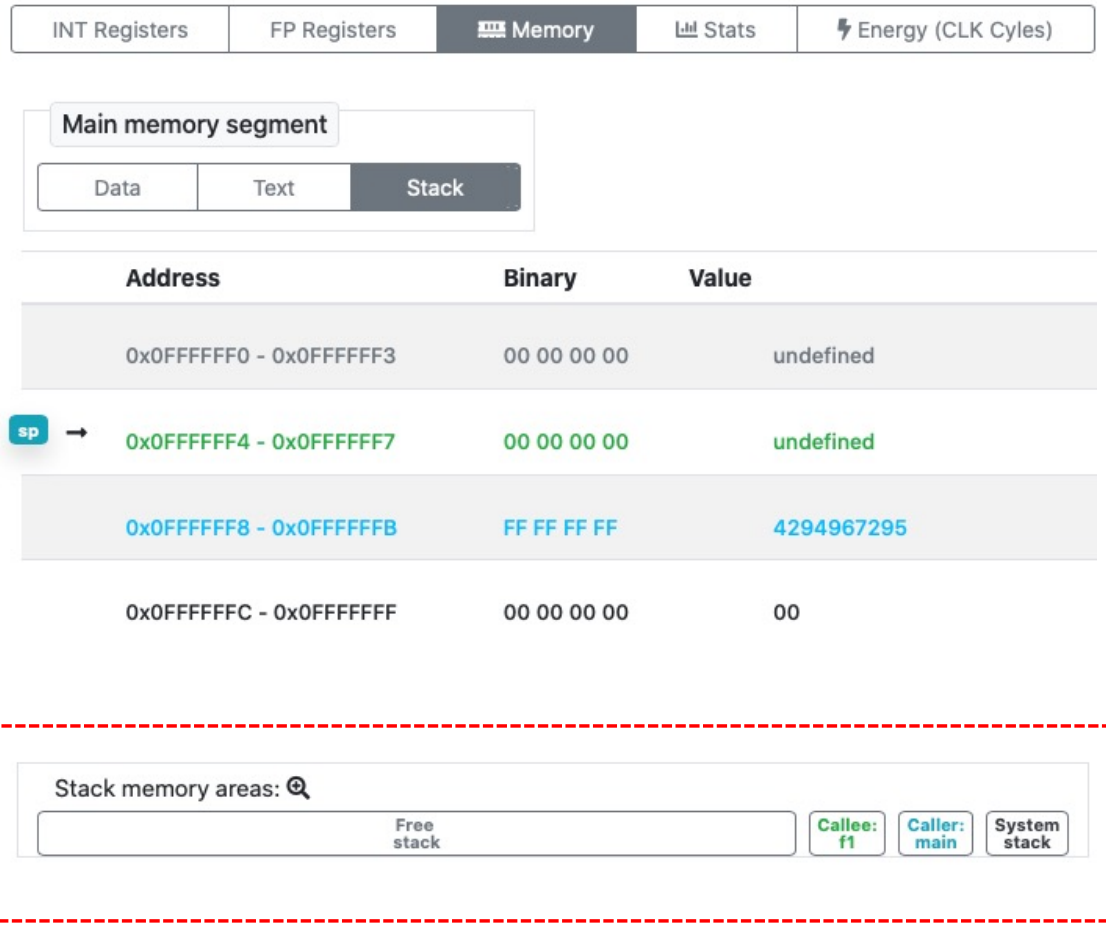
Ejecución del programa. Bucles



Ejemplo de llamadas a funciones anidadas

ejemplo

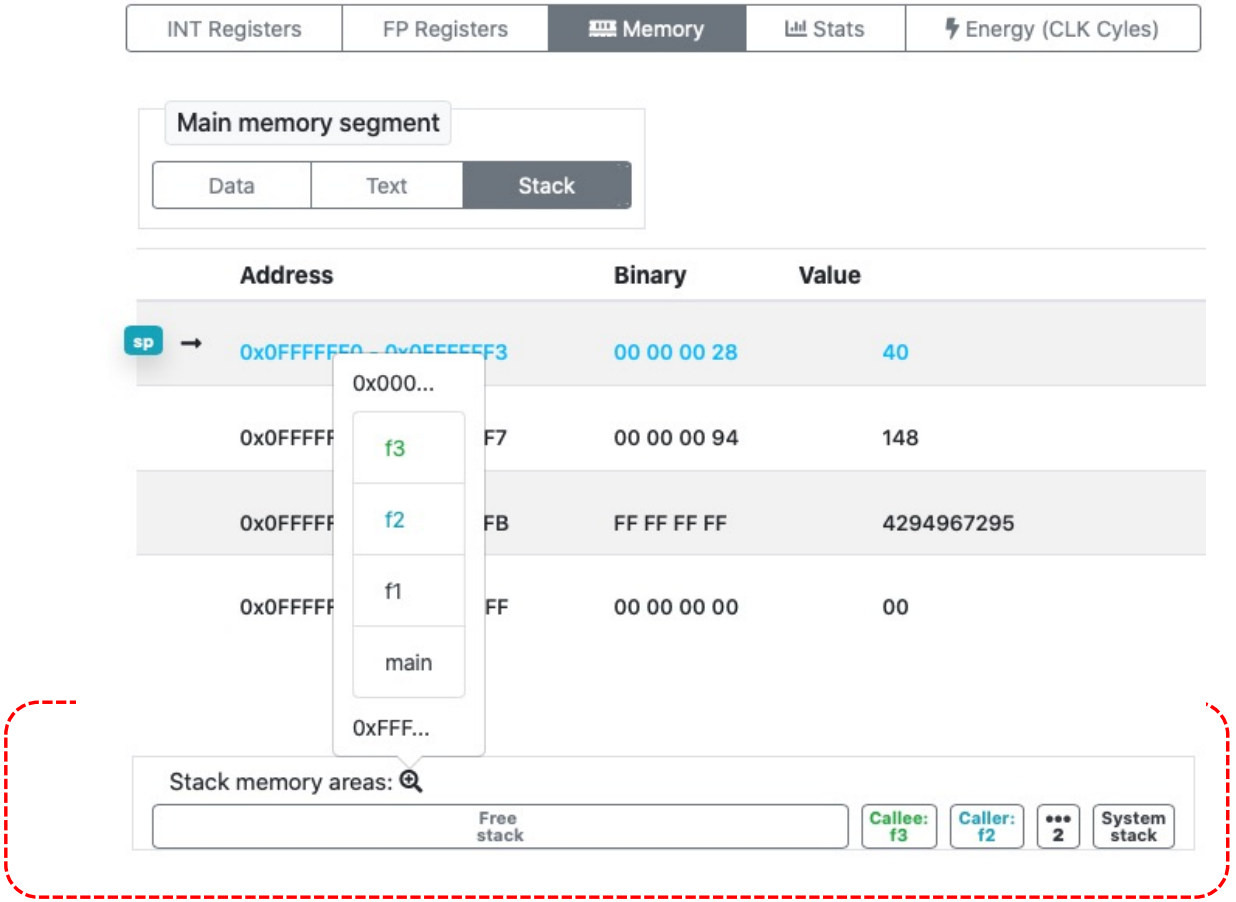
- Comprobar el crecimiento de la pila



Ejemplo de llamadas a funciones anidadas

ejemplo

- Comprobar el crecimiento de la pila



Llamadas a funciones

- Convenio simplificado
  - La pila no necesita estar alineada a 8 bytes
- Alerta si se incumple el convenio de paso de parámetros y uso de pila

Integer Registers	
Register Name	**Usage**
zero	Constant 0
ra	Return address (routines/functions)
sp	Stack pointer
gp	Global pointer
tp	Thread pointer
t0..t6	Temporary (NOT preserved across calls)
s0..s11	Saved temporary (preserved across calls)
a0, a1	Arguments for functions / return value
a2..a7	Arguments for functions
Floating-point registers	
ft0..ft11	Temporary (NOT preserved across calls)
fs0..fs11	Saved temporary (preserved across calls)
fa0, fa1	Arguments for functions / return value
fa2..fa7	Arguments for functions

Detección de errores en el convenio de paso de parámetros

- Corregir los fallos que aparecen en el ejemplo por un uso incorrecto del convenio de paso de parámetros y uso de pila
- Modelo simplificado que se utiliza actualmente
  - La pila no tiene porqué estar alineada a 8

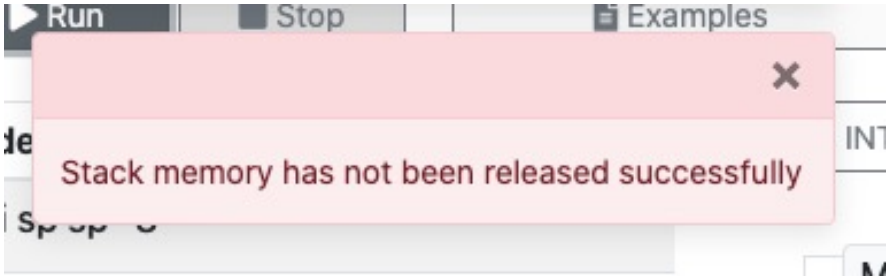
Integer Registers	
Register Name	**Usage**
zero	Constant 0
ra	Return address (routines/functions)
sp	Stack pointer
gp	Global pointer
tp	Thread pointer
t0..t6	Temporary (NOT preserved across calls)
s0..s11	Saved temporary (preserved across calls)
a0, a1	Arguments for functions / return value
a2..a7	Arguments for functions
Floating-point registers	
ft0..ft11	Temporary (NOT preserved across calls)
fs0..fs11	Saved temporary (preserved across calls)
fa0, fa1	Arguments for functions / return value
fa2..fa7	Arguments for functions

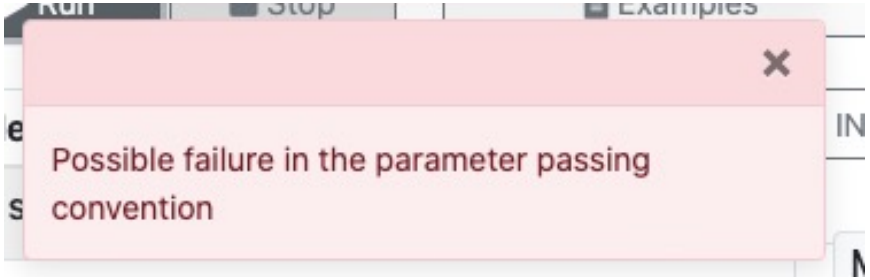
ejemplo

```
.data

.text

    max:            addi    sp, sp -8
                    sw      s0, 0(sp)
                    mv      s0, a0
                    mv      s1, a1
                    bge     s0, s1, bigger
                    mv      a0, s1
                    jr      ra
            bigger: mv      a0, s0
                    jr      ra


    main:           li      a0, 8
                    li      a1, 7
                    jal     ra, max

                    li      a7, 1
                    ecall

                    li a7, 10
                    ecall
```

Detección de errores en el convenio de paso de parámetros

- Fallos detectados:



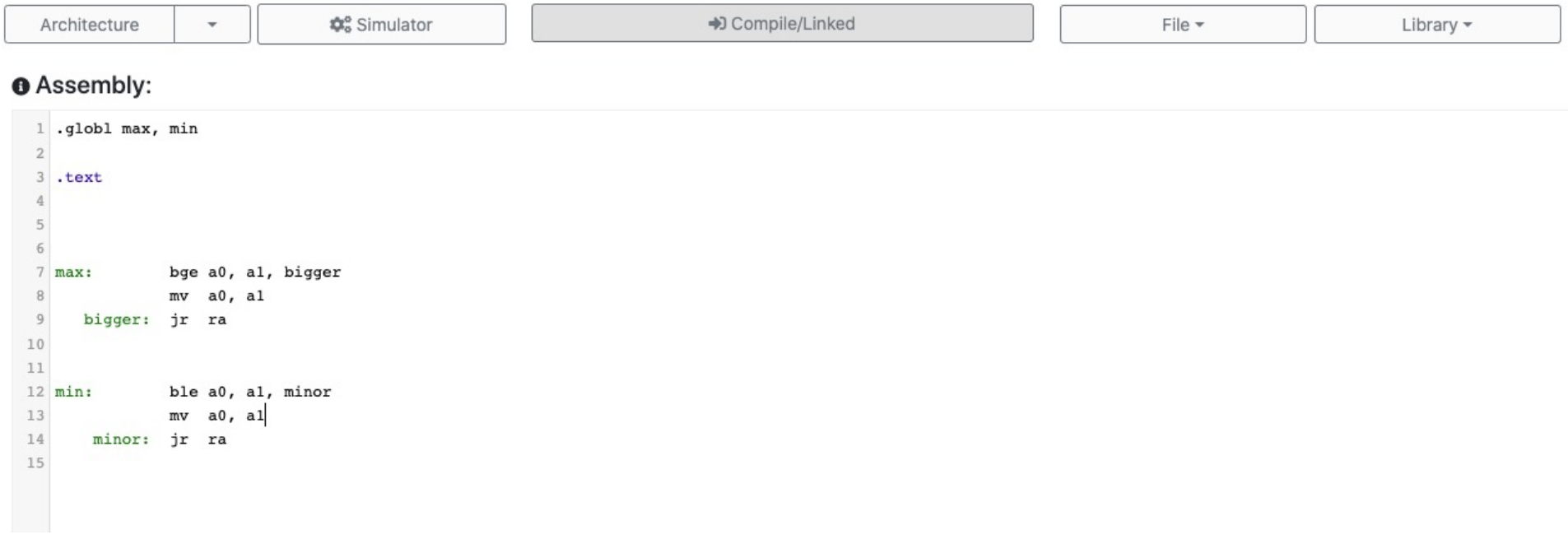


ejemplo

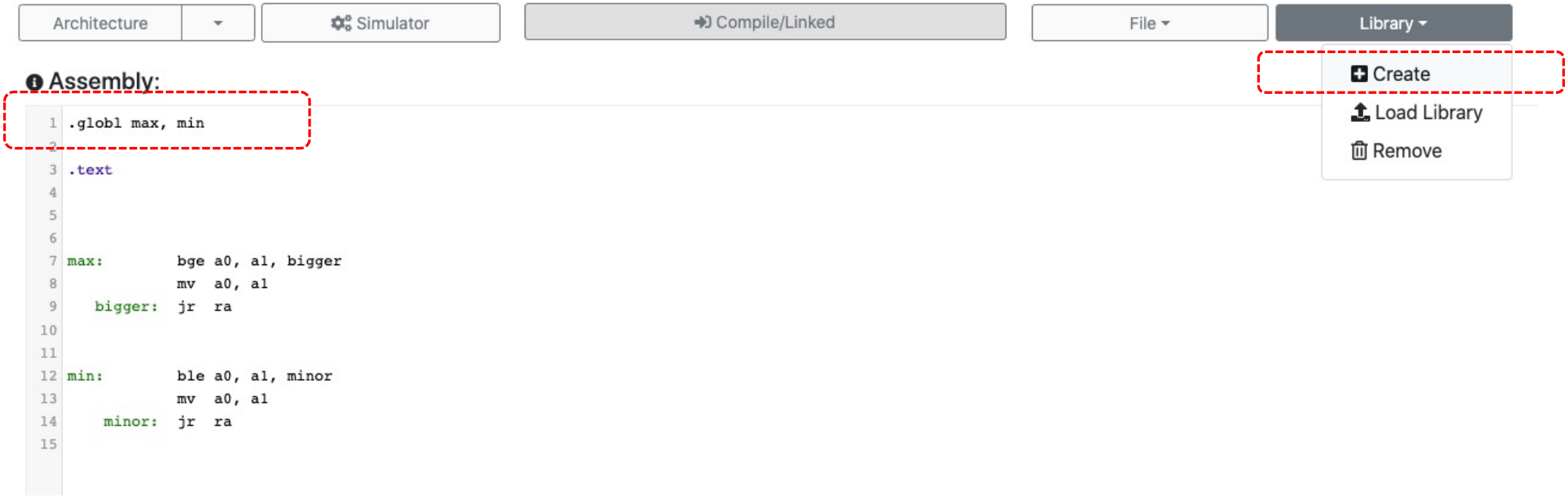
```
.data

.text

    max:            addi    sp, sp -8
                    sw      s0, 0(sp)
                    mv      s0, a0
                    mv      s1, a1
                    bge     s0, s1, bigger
                    mv      a0, s1
                    jr      ra
            bigger: mv      a0, s0
                    jr      ra


    main:           li      a0, 8
                    li      a1, 7
                    jal     ra, max

                    li      a7, 1
                    ecall

                    li a7, 10
                    ecall
```

Creación de librerías

ejemplo

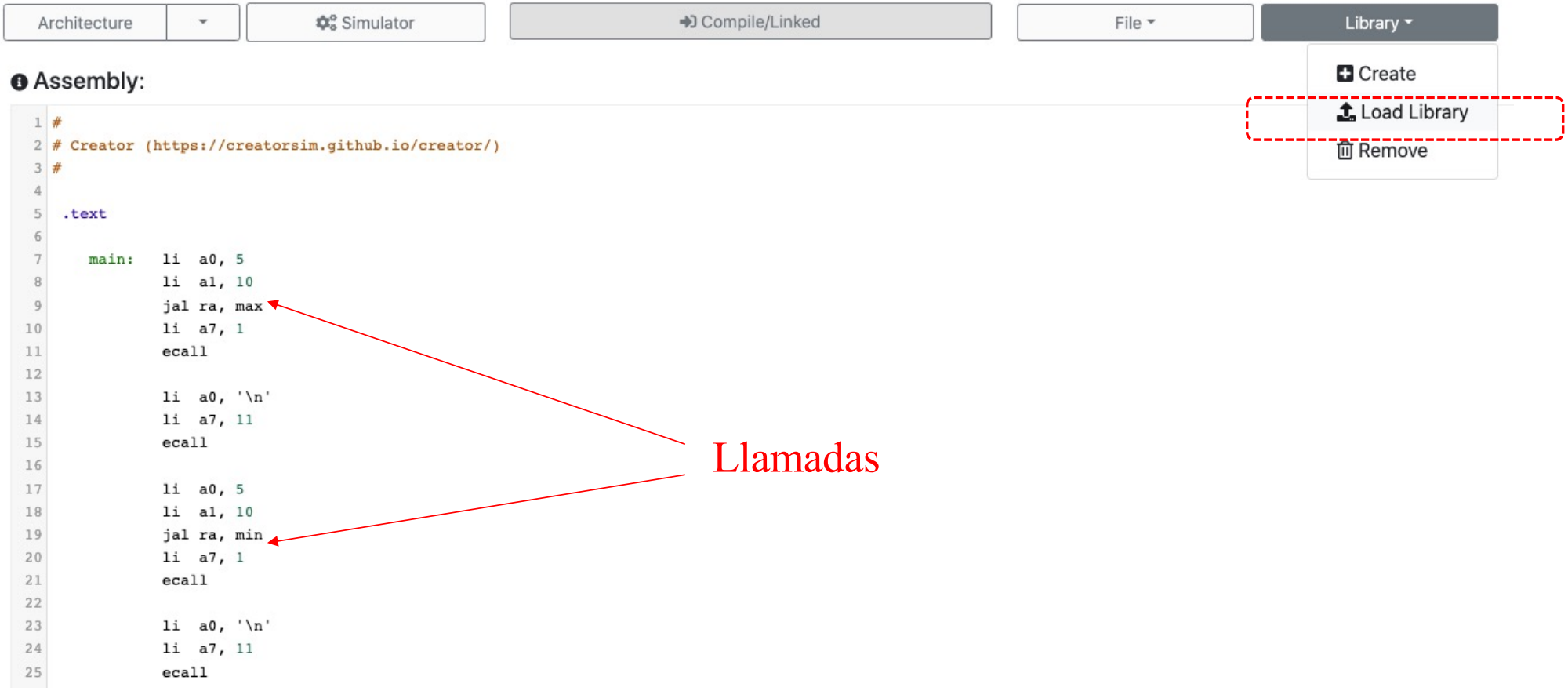


Creación de librerías

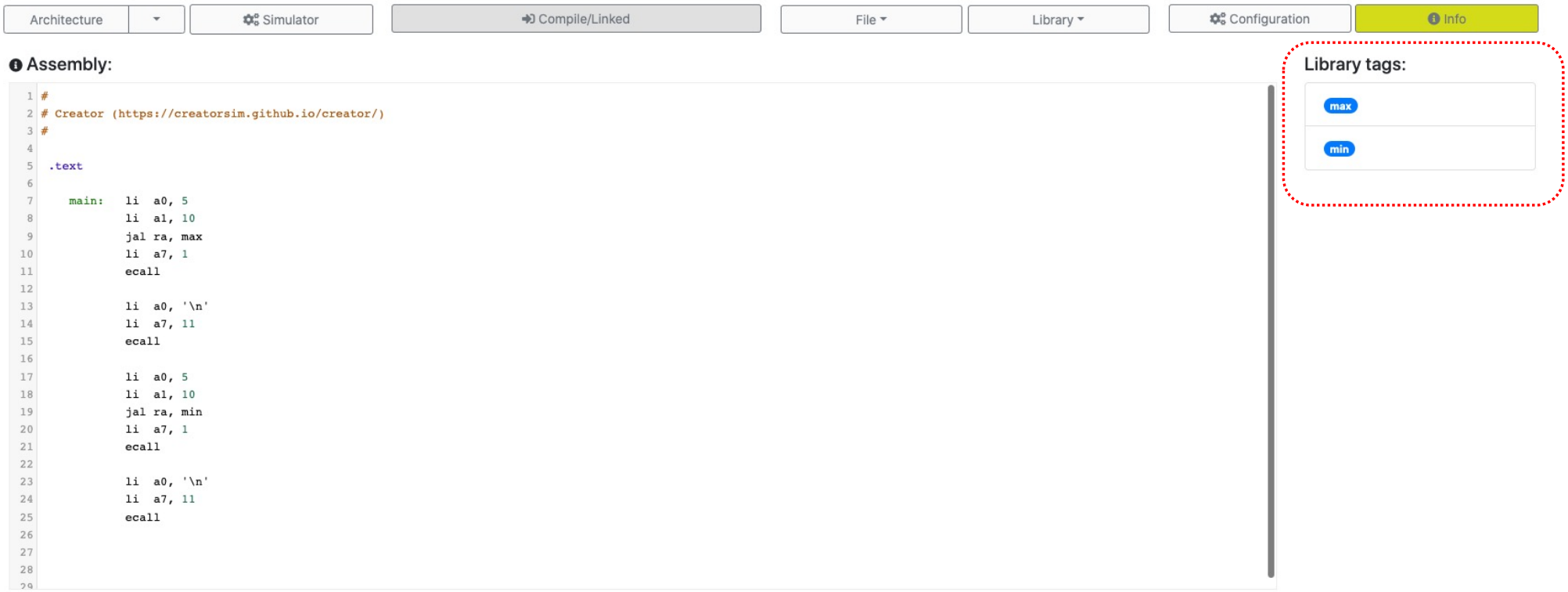


Uso de librerías

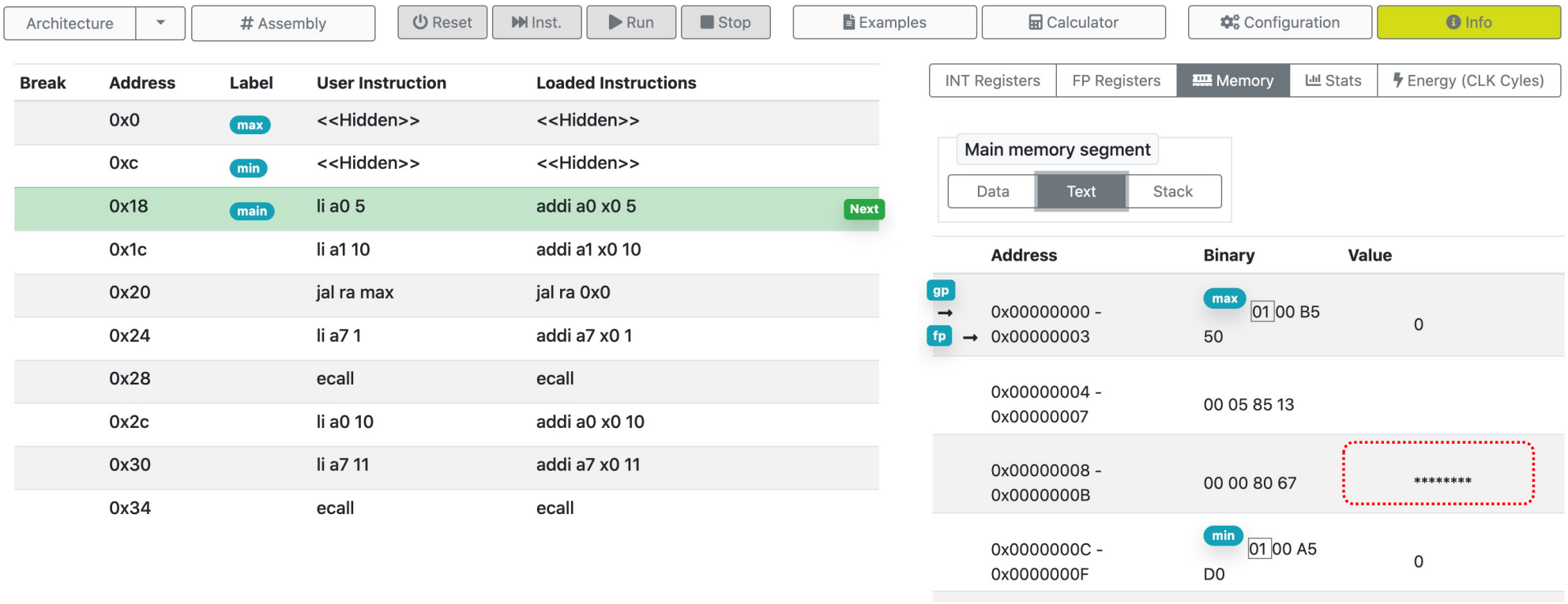
ejemplo



Librería cargada



Uso de librerías

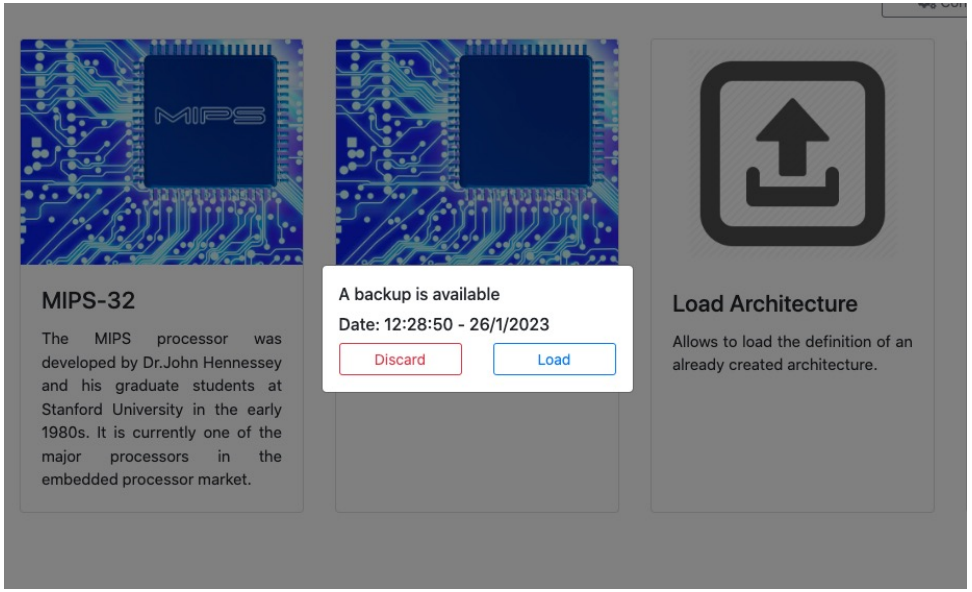


Errores en tiempo de ejecución

- Programa sin función main
- Puntero de pila (sp) apunta al segmento de datos o de texto
- Escritura en el segmento de texto
- Acceso a una posición de memoria no alineada

Caché del navegador para recuperación de errores

- El programa que se está editando se guarda en la caché del navegador cada vez que se compila
- Si el navegador falla se puede recuperar el programa al volverlo a cargar



Contenido (RISC-V)

- Juego de instrucciones soportado
- Visión del estudiante:
  - Características del entorno
  - Edición y compilación de programas
  - Ejecución y depuración de programas
  - Bibliotecas de funciones
  - Facilidades para entender el empleo de funciones y uso de pila
- Visión del profesor:
  - Soporte a la corrección de prácticas
  - Soporte a la creación de material didáctico
  - Capacidades para extender el juego de instrucciones y crear nuevas arquitecturas

Soporte a la corrección de prácticas

- Ejecución en línea de comandos
- Prerrequisitos:
  - Linux, node.js y npm
- Pasos:
  - Descargar el repositorio:
    - `git clone https://github.com/creatorsim/creator.git`
  - `cd creator`
  - Instalar los paquetes:
    - `npm install terser jshint colors yargs readline-sync`

Compilación y ejecución de un programa

- `./creator.sh -h`

```
CREATOR
-------
version: 3.2
website: https://creatorsim.github.io/

Usage: creator.sh -a <file name> -s <file name>
Usage: creator.sh -h

Options:
      --version       Show version number                                [boolean]
  -a, --architecture  Architecture file        [string] [required] [default: ""]
  -s, --assembly      Assembly file            [string] [required] [default: ""]
  -d, --directory     Assemblies directory                [string] [default: ""]
  -l, --library       Assembly library file               [string] [default: ""]
  -r, --result        Result file to compare with         [string] [default: ""]
      --describe      Help on element                     [string] [default: ""]
      --maxins        Maximum number of instructions to be executed
                                                 [string] [default: "1000000"]
  -o, --output        Define output format        [string] [default: "normal"]
      --color         Colored output                [boolean] [default: false]
  -h, --help          Show help                                          [boolean]

Examples:
  ./creator.sh  To show examples._
```

Ejecución de un programa

- `./creator.sh -a architecture/RISC_V_RV32IMFD.json -s ./factorial.s`

```
CREATOR
-------
version: 3.2
website: https://creatorsim.github.io/

[./factorial.s]
120
[Architecture] Architecture 'architecture/RISC_V_RV32IMFD.json' loaded successfully.
[Library] Without library
[Compile] Code './factorial.s' compiled successfully.
[Execute] Executed successfully.
[FinalState] cr[PC]:0x18; ir[ra,x1]:0x8; ir[t0,x5]:0x2; ir[t1,x6]:0x5; ir[a0,x10]:0x78; ir[a7,x17]:0xa; keyboard[0x0]:'';
display[0x0]:'120';
```

Ejecución de un programa y comprobación de resultados

- Podemos comparar la salida con un resultado de referencia
- Ejemplo de resultado de referencia para comparar solo la salida:

referencia.txt

```
display[0x0]:'120';
```

- Ejecución y comparación con salida de referencia:
  - `./creator.sh -a architecture/RISC_V_RV32IMFD.json -s ./factorial.s -r referencia.txt`

Ejemplo de salida correcta

- `./creator.sh -a architecture/ RISC_V_RV32IMFD.json  -s factorial.s -r referencia.txt`

```
CREATOR
-------
version: 3.2
website: https://creatorsim.github.io/

[./factorial.s]
120
[Architecture] Architecture 'architecture/RISC_V_RV32IMFD.json' loaded successfully.
[Library] Without library
[Compile] Code './factorial.s' compiled successfully.
[Execute] Executed successfully.
[State] Equals
```

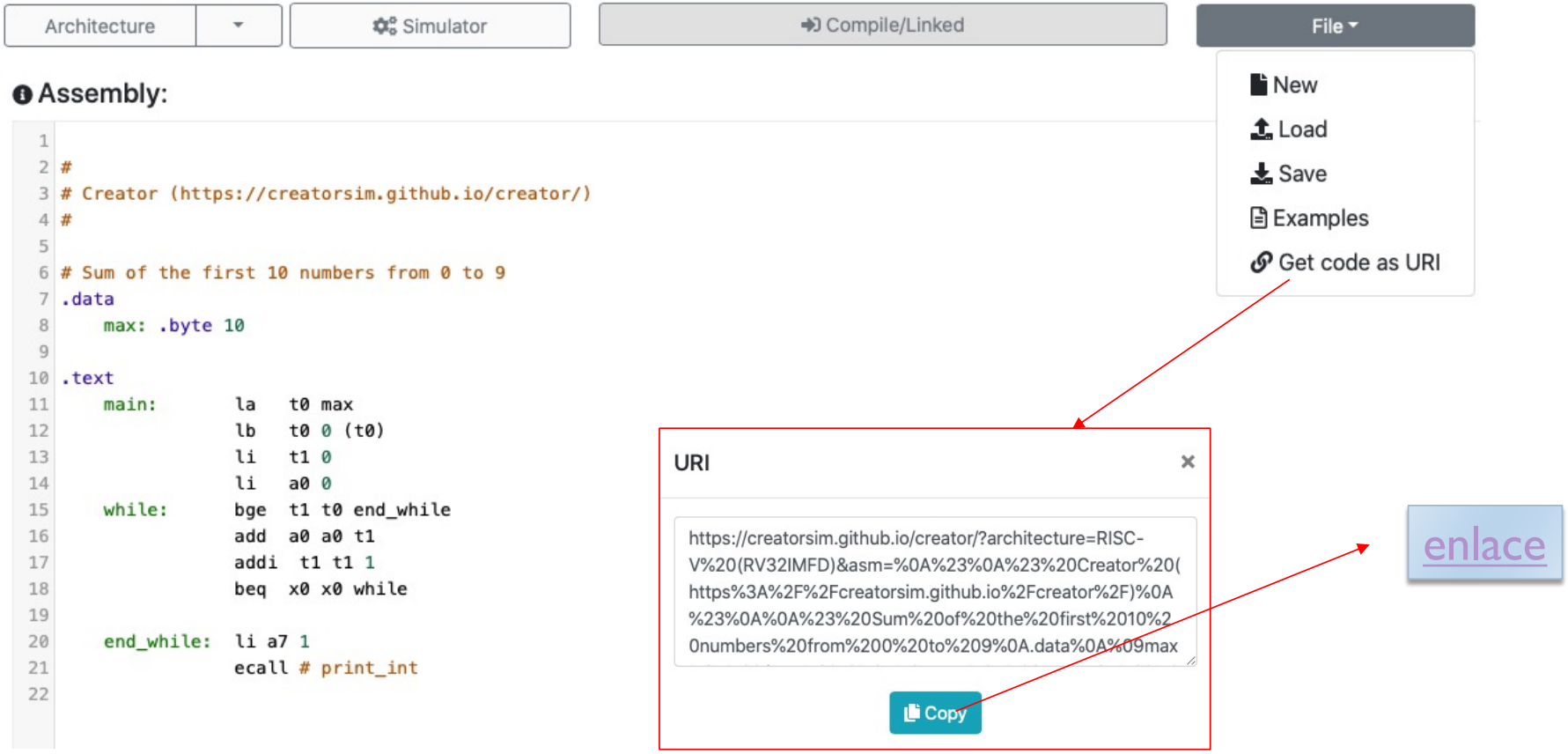
Ejemplo de salida incorrecta

- `./creator.sh -a architecture/ RISC_V_RV32IMFD.json -s factorial.s -r referencia.txt`

```
CREATOR
-------
version: 3.2
website: https://creatorsim.github.io/

[./factorial.s]
808
[Architecture] Architecture 'architecture/RISC_V_RV32IMFD.json' loaded successfully.
[Library] Without library
[Compile] Code './factorial.s' compiled successfully.
[Execute] Executed successfully.
[State] Different: display[0x0]='120' is ='808'.
```

Ayuda a la creación de materiales docentes

Ayuda a la creación de materiales docentes



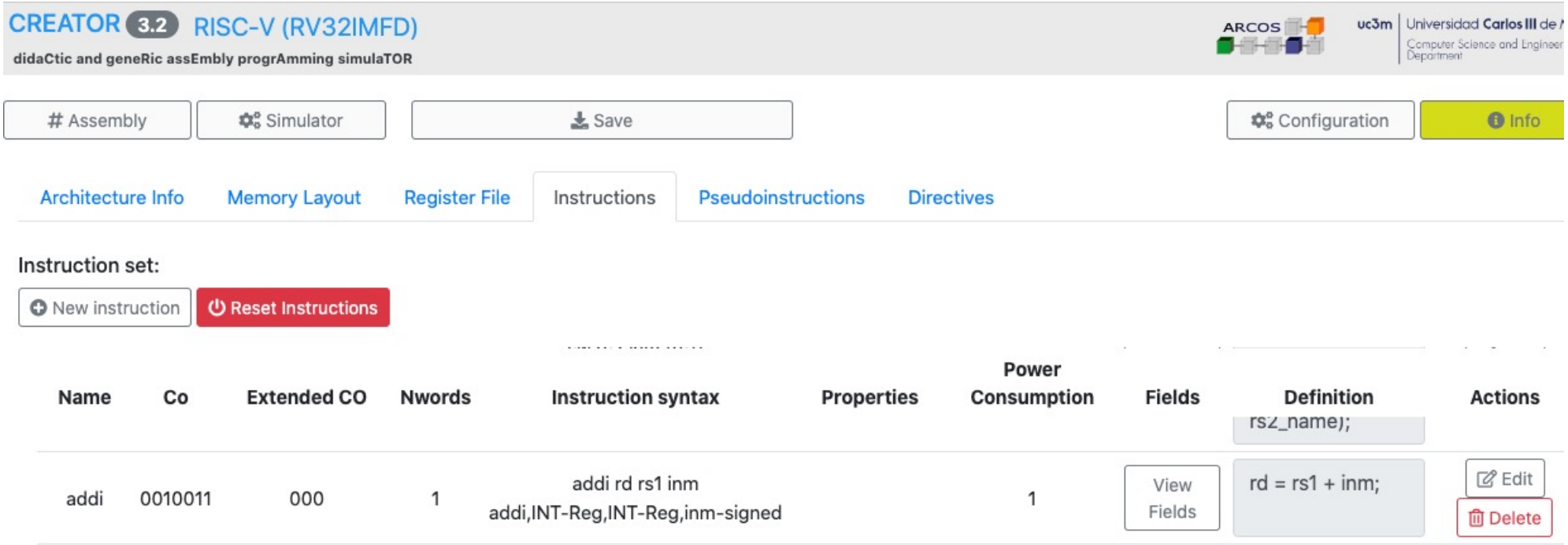
Capacidades para extender el juego de instrucciones y crear nuevas arquitecturas

Architecture Info | Memory Layout | Register File | Instructions | Pseudoinstructions | Directives

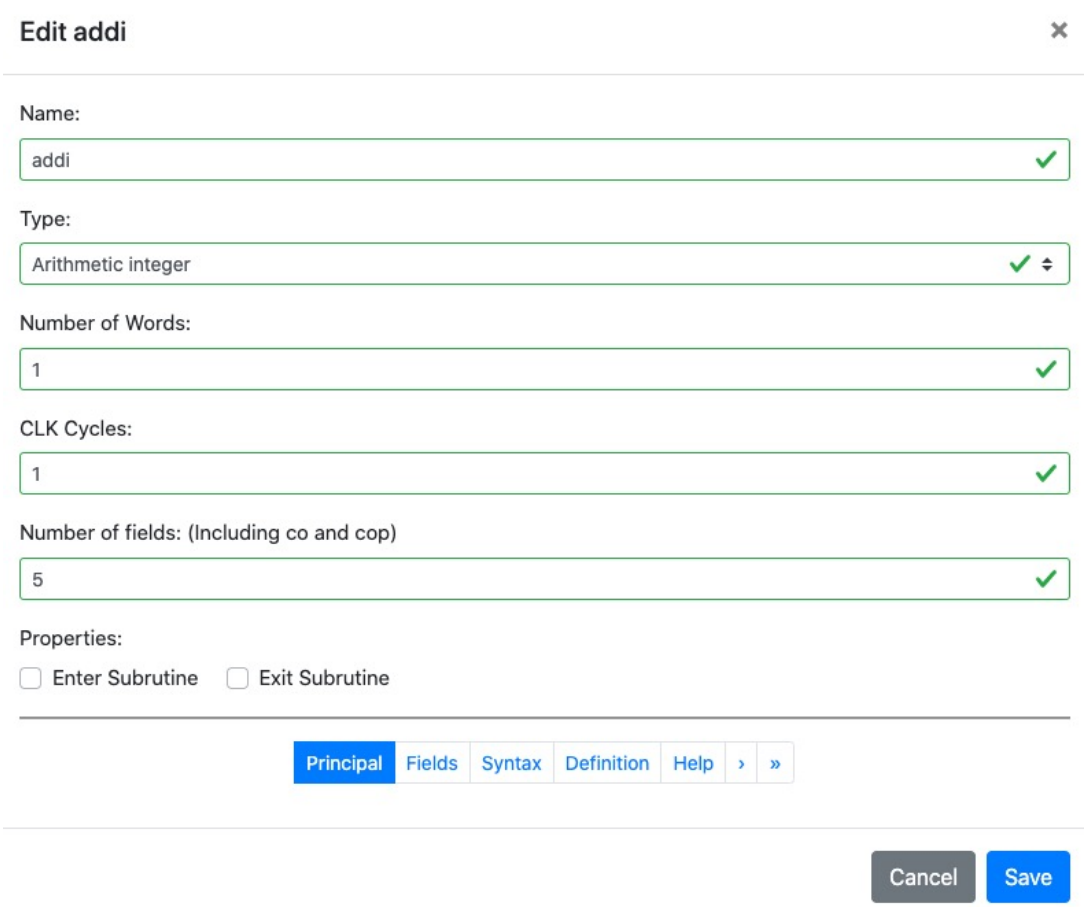
Architecture general information:

Field	Value	Actions
Name	RISC-V (RV32IMFD)	
Bits	32	Edit Reset
Data Format	Big Endian	Edit Reset
Memory Alignment	Enabled	Edit Reset
Main Function	main	Edit Reset
Passing Convention	Enabled	Edit Reset
Sensitive Register Name	Enabled	Edit Reset

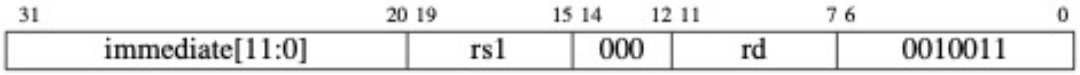
Ejemplo de definición de una instrucción (addi)



Ejemplo de definición de una instrucción (addi)



Ejemplo de definición de una instrucción (addi)



Edit addi

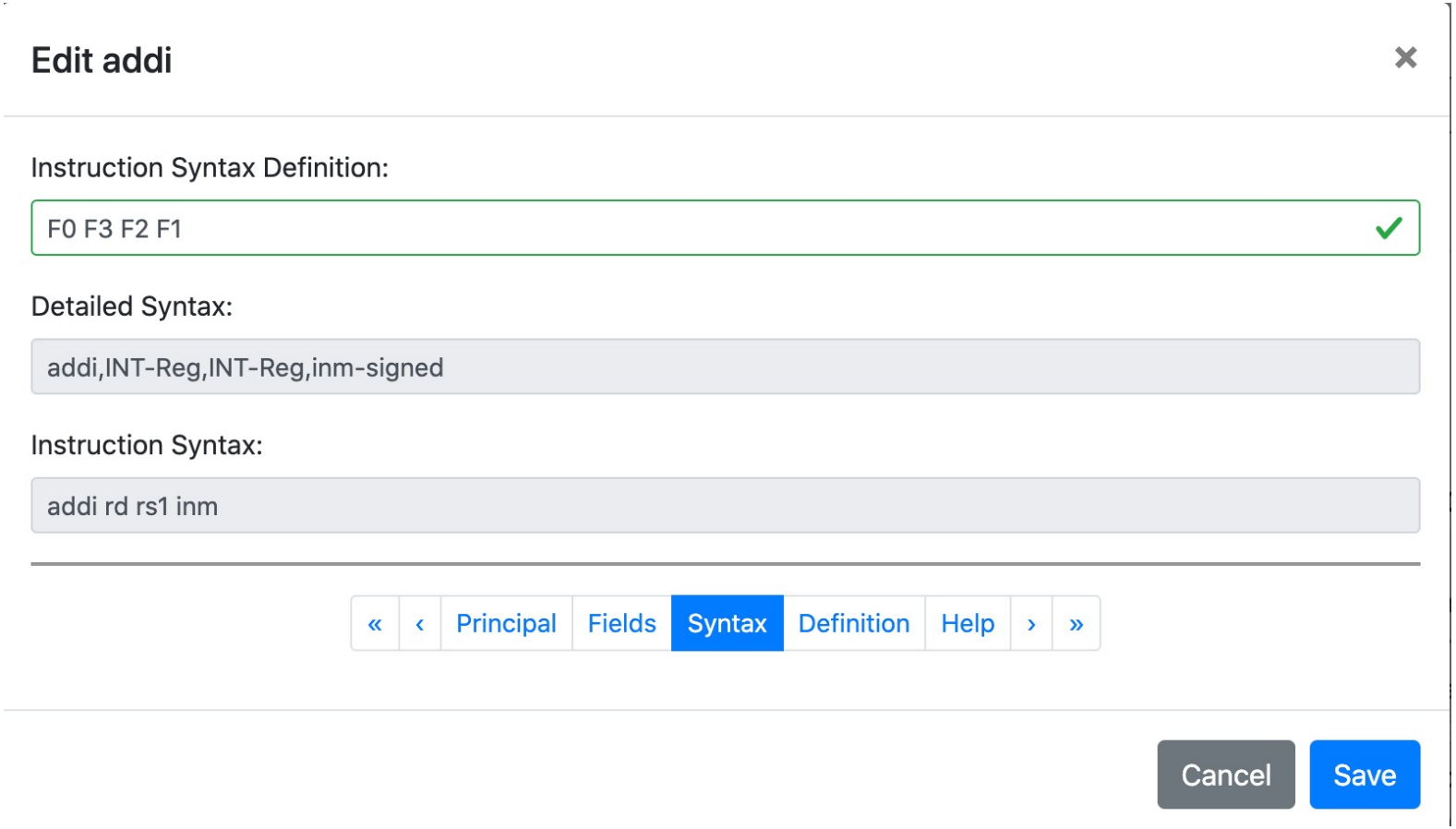
	Name:	Type	Break	Start Bit	End Bit	
Field 0	addi	co		6	0	0010011
Field 1	inm	inm-sign		31	20	
Field 2	rs1	INT-Reg		19	15	
Field 3	rd	INT-Reg		11	7	
Field 4	cop	cop		14	12	000

código de operación

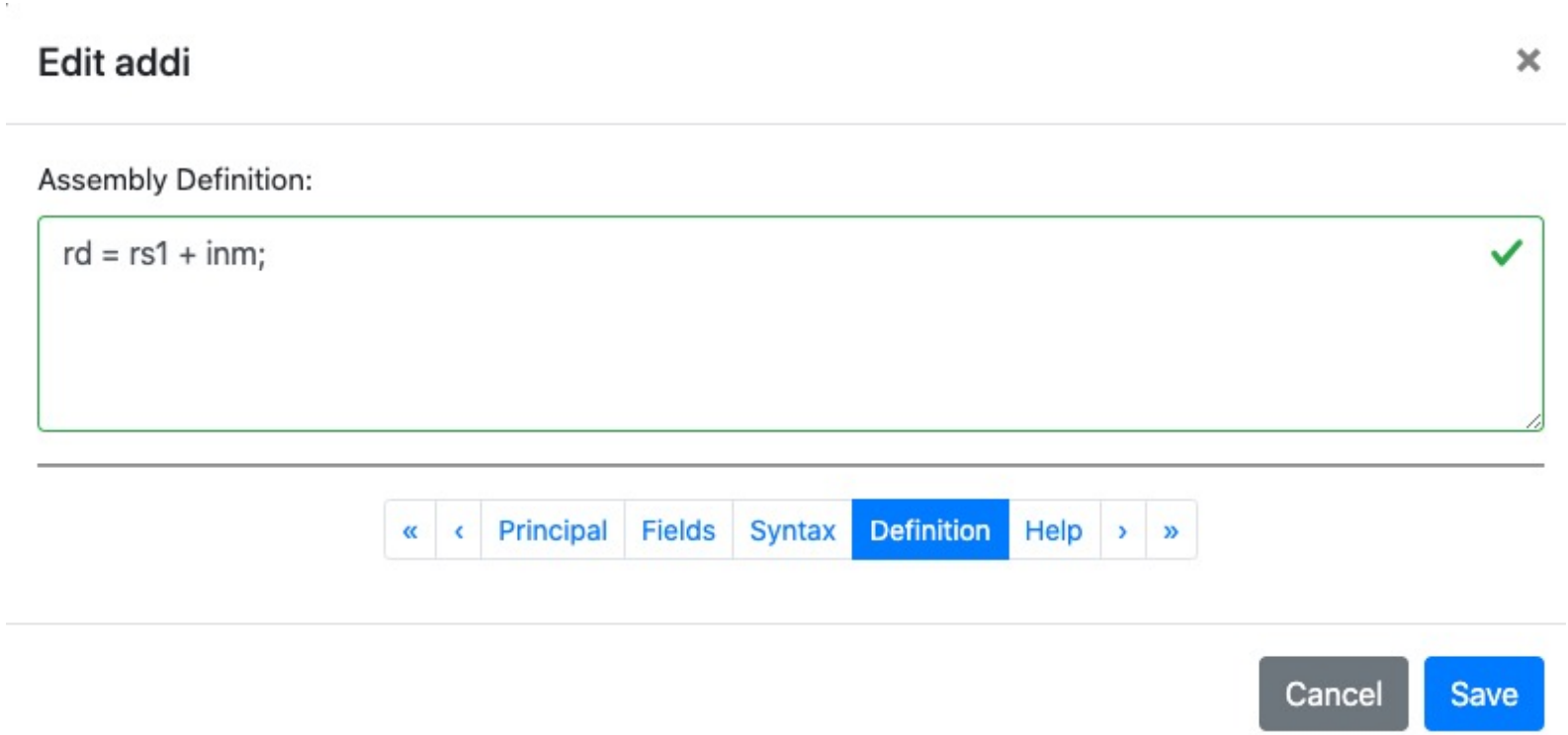
« ‹ Principal Fields Syntax Definition Help › »

Cancel Save

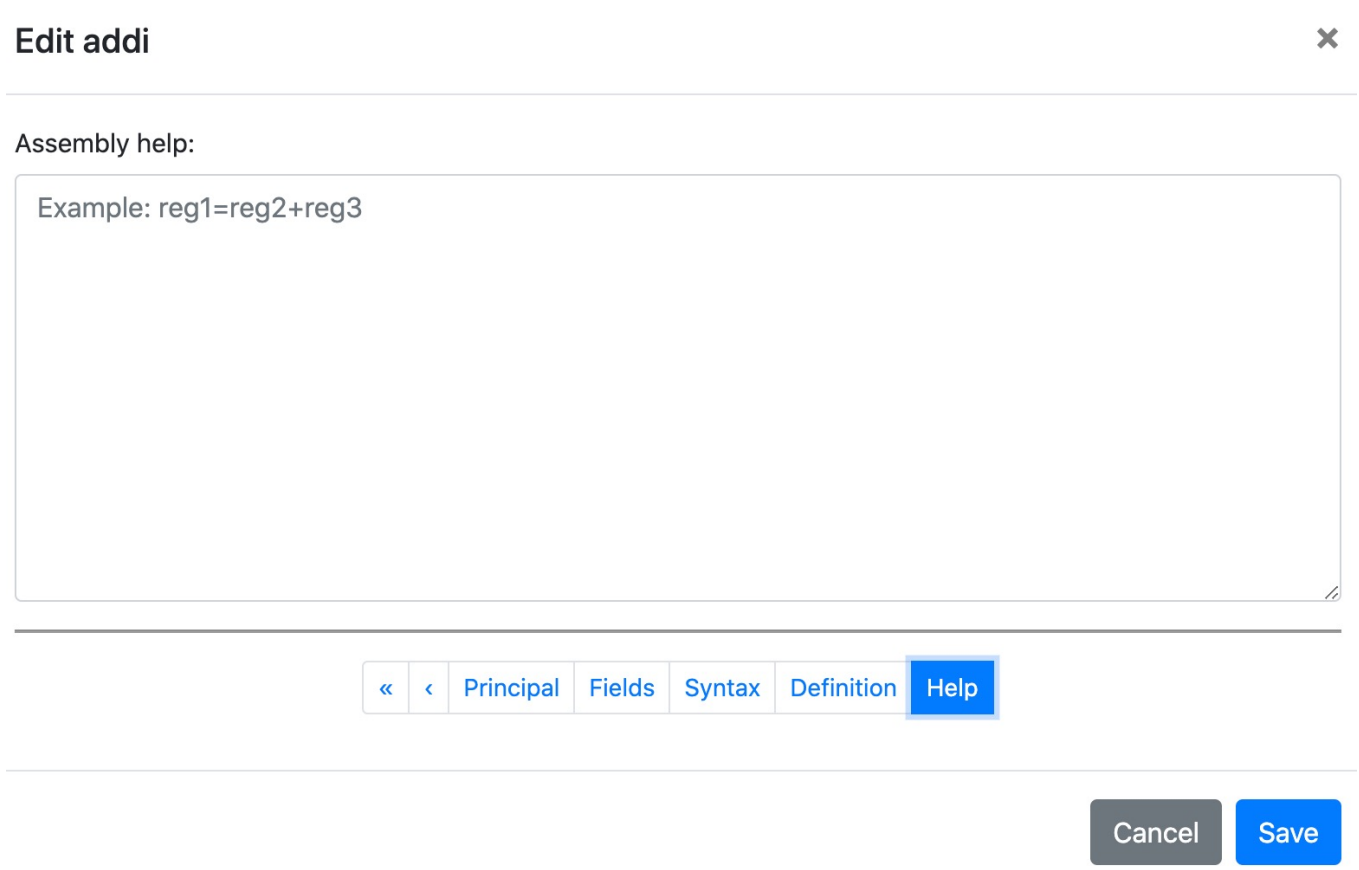
Ejemplo de definición de una instrucción (addi)



Ejemplo de definición de una instrucción (addi)

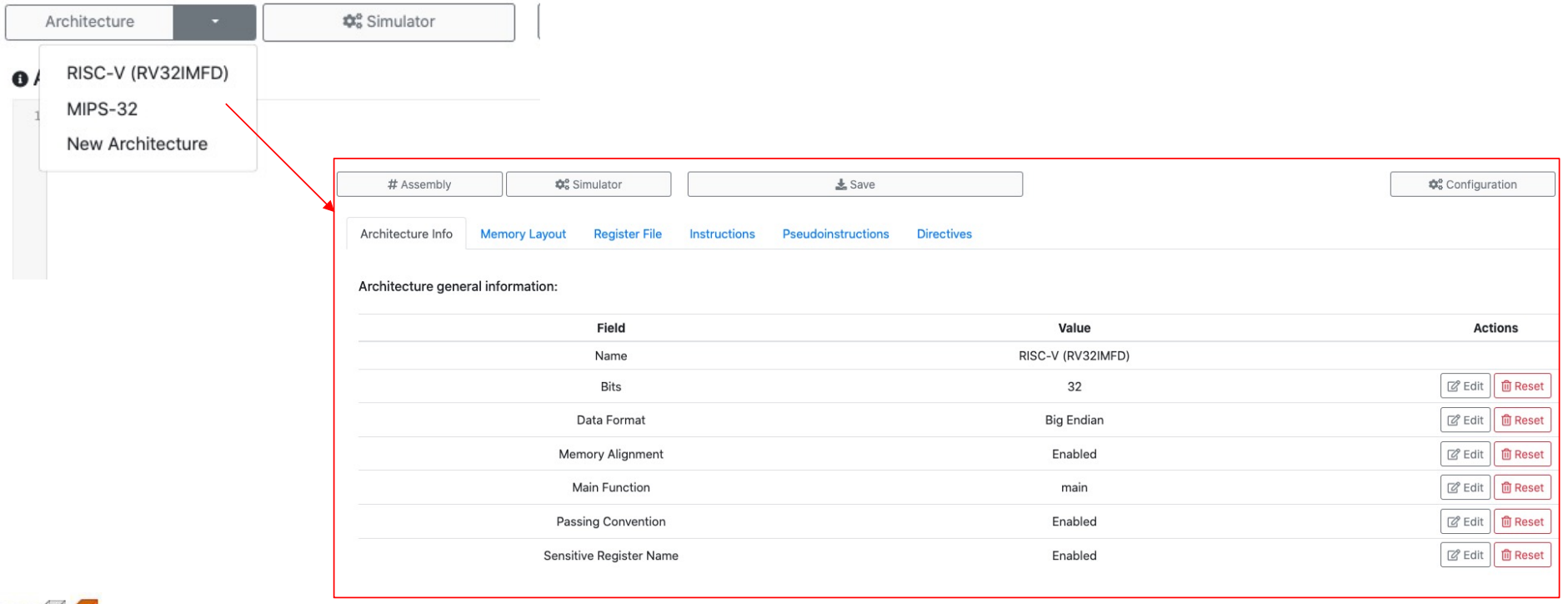


Ejemplo de definición de una instrucción (addi)

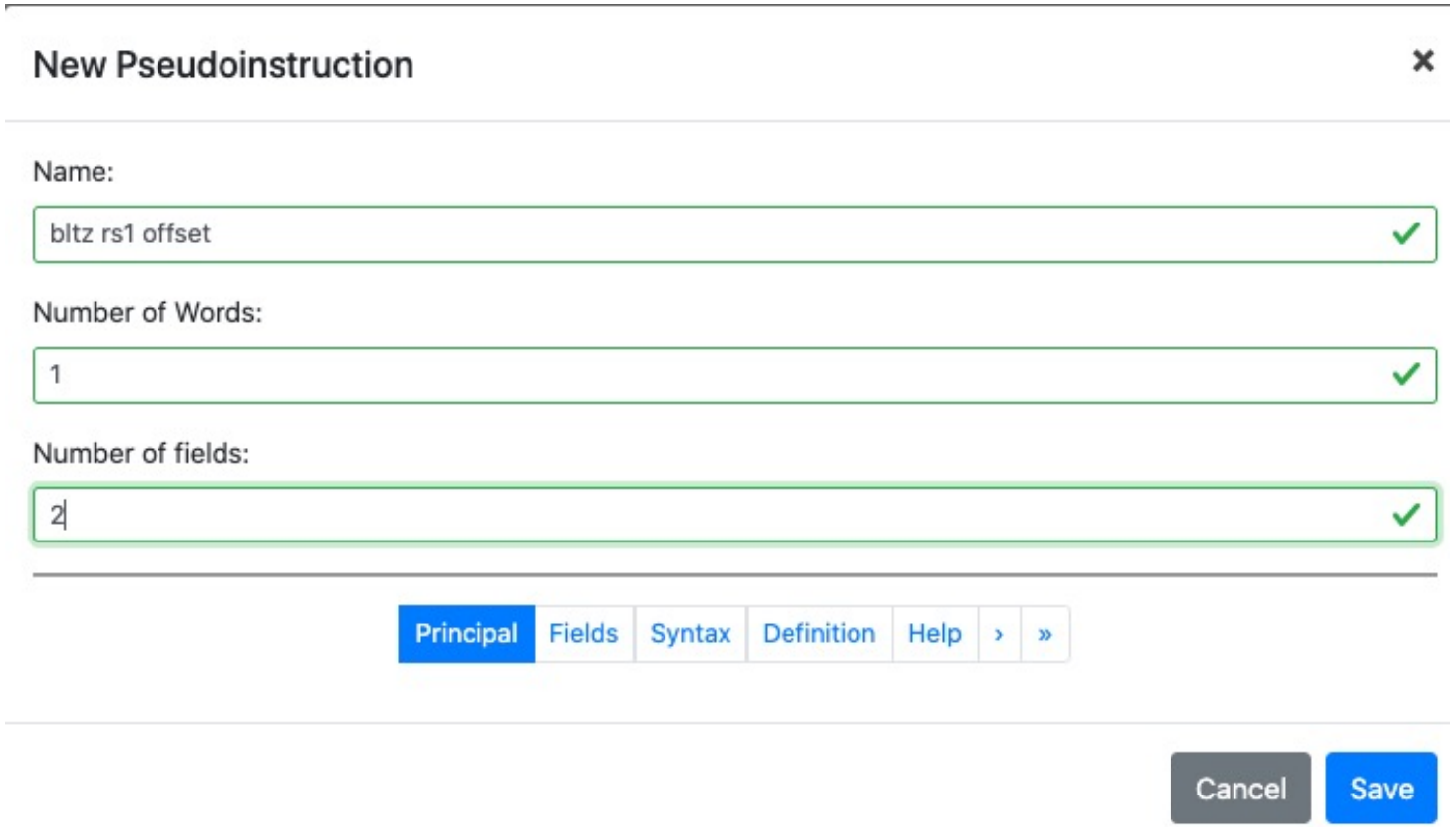


Creación de una nueva pseudoinstrucción

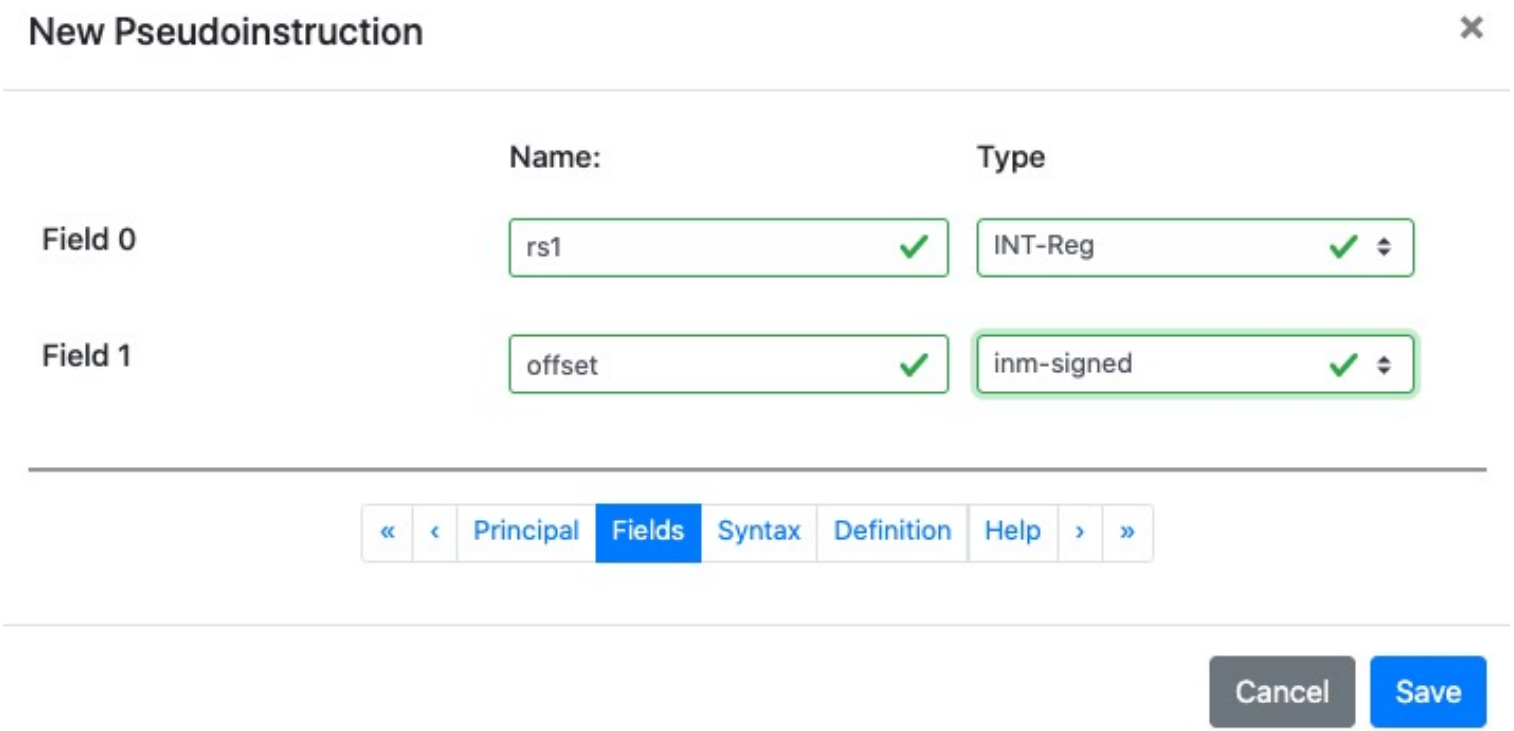
- Ejemplo:
  - `bltz  rs1, offset` `if (rs1 < 0)  PC = PC + offset`
  - Se expande a: `blt rs1, zero, offset`



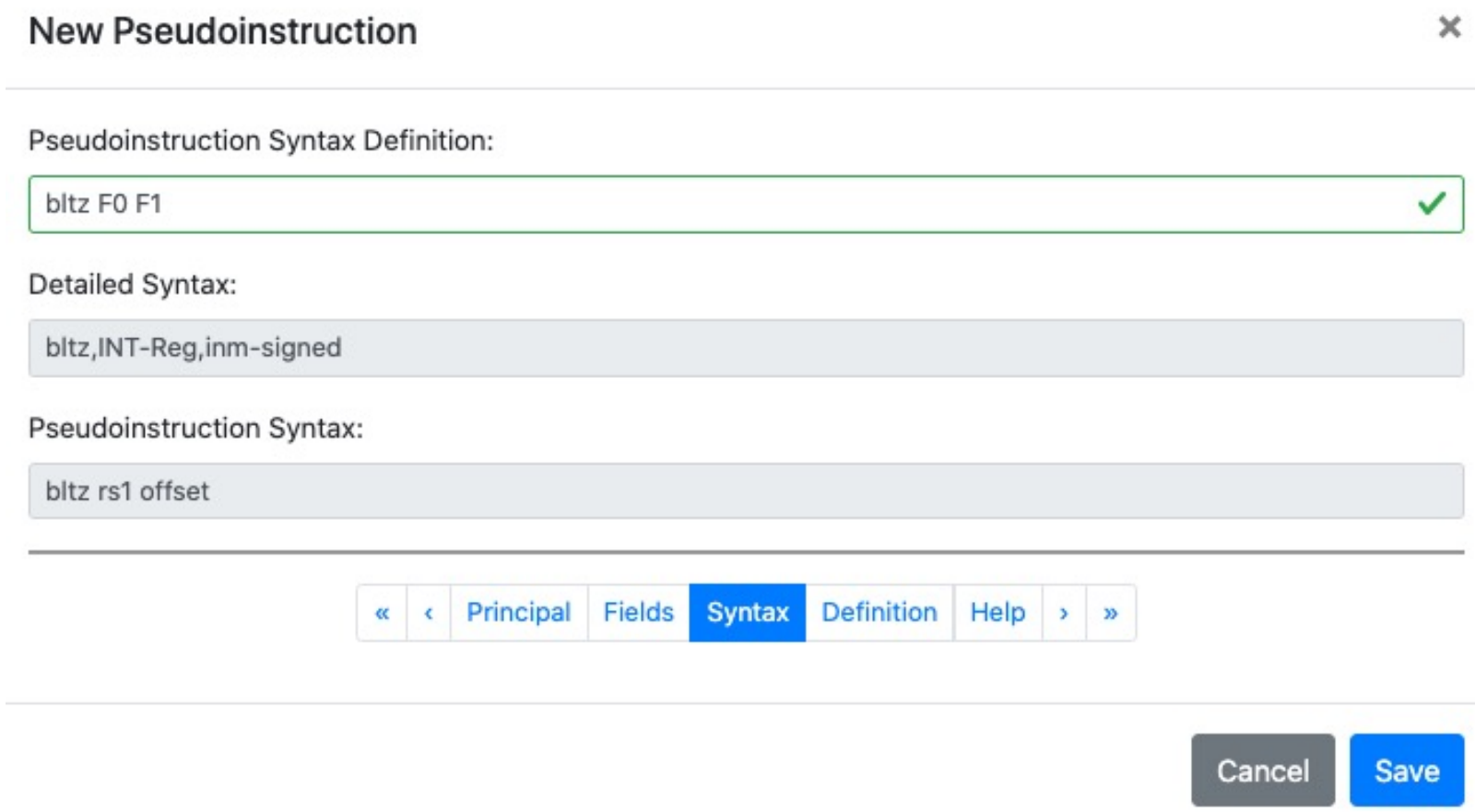
Creación de una nueva pseudoinstrucción



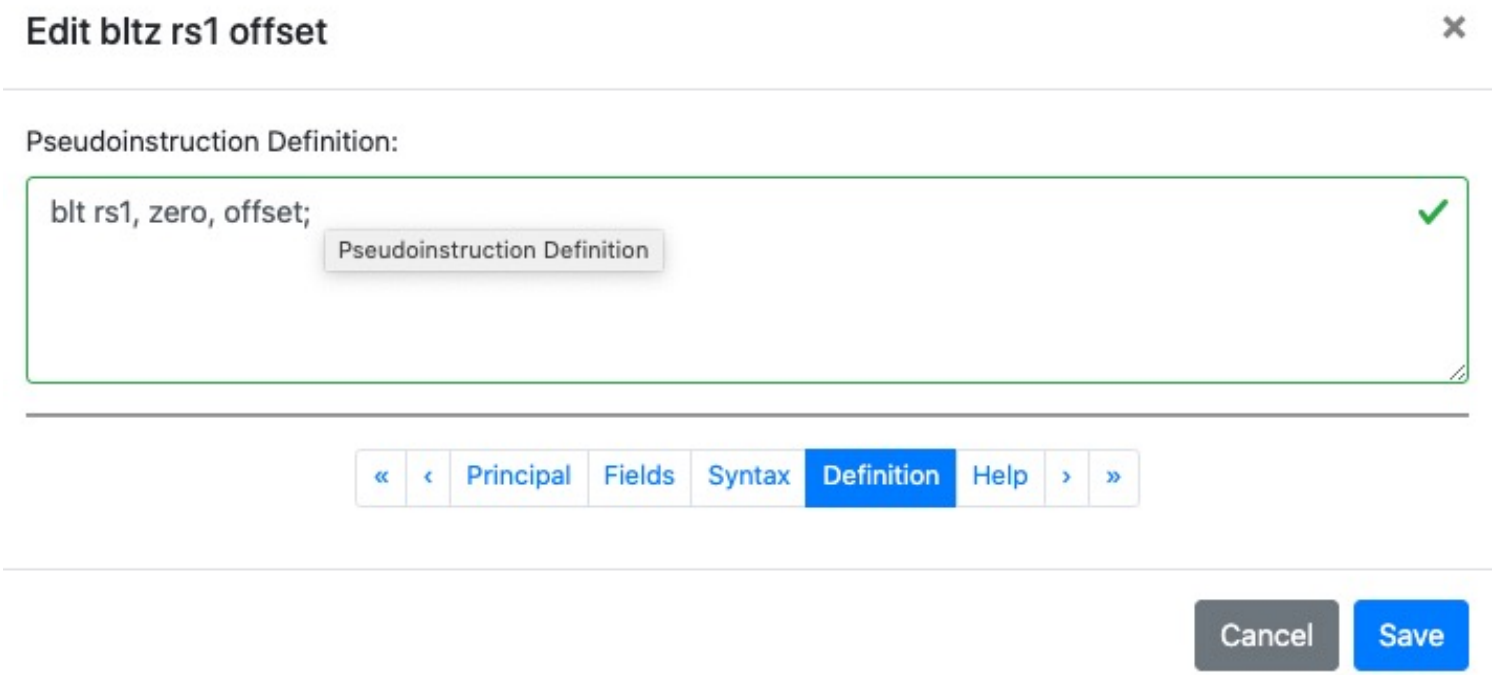
Creación de una nueva pseudoinstrucción



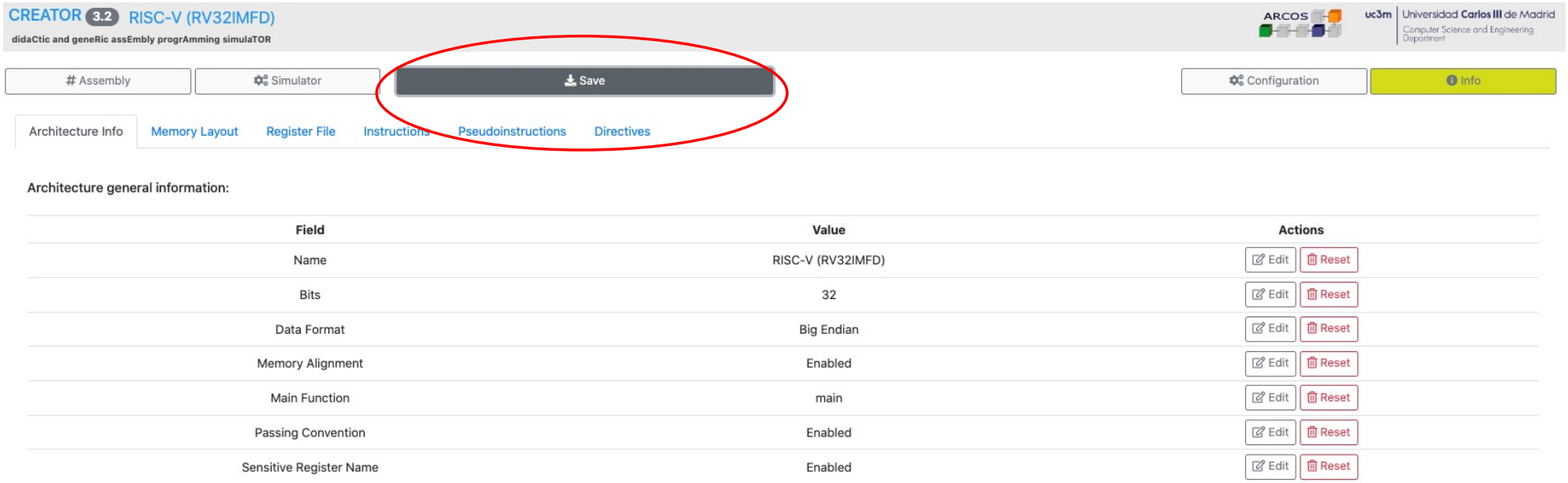
Creación de una nueva pseudoinstrucción



Creación de una nueva pseudoinstrucción



Creación de una nueva pseudoinstrucción



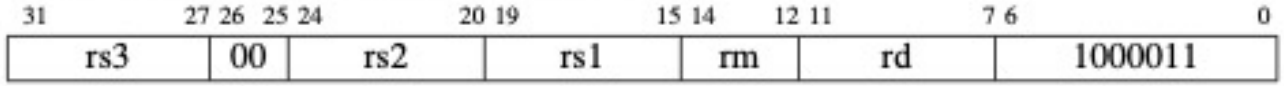
Creación de una nueva instrucción

- Ejemplo: `fmadd.s rd, rs1, rs2, rs3`

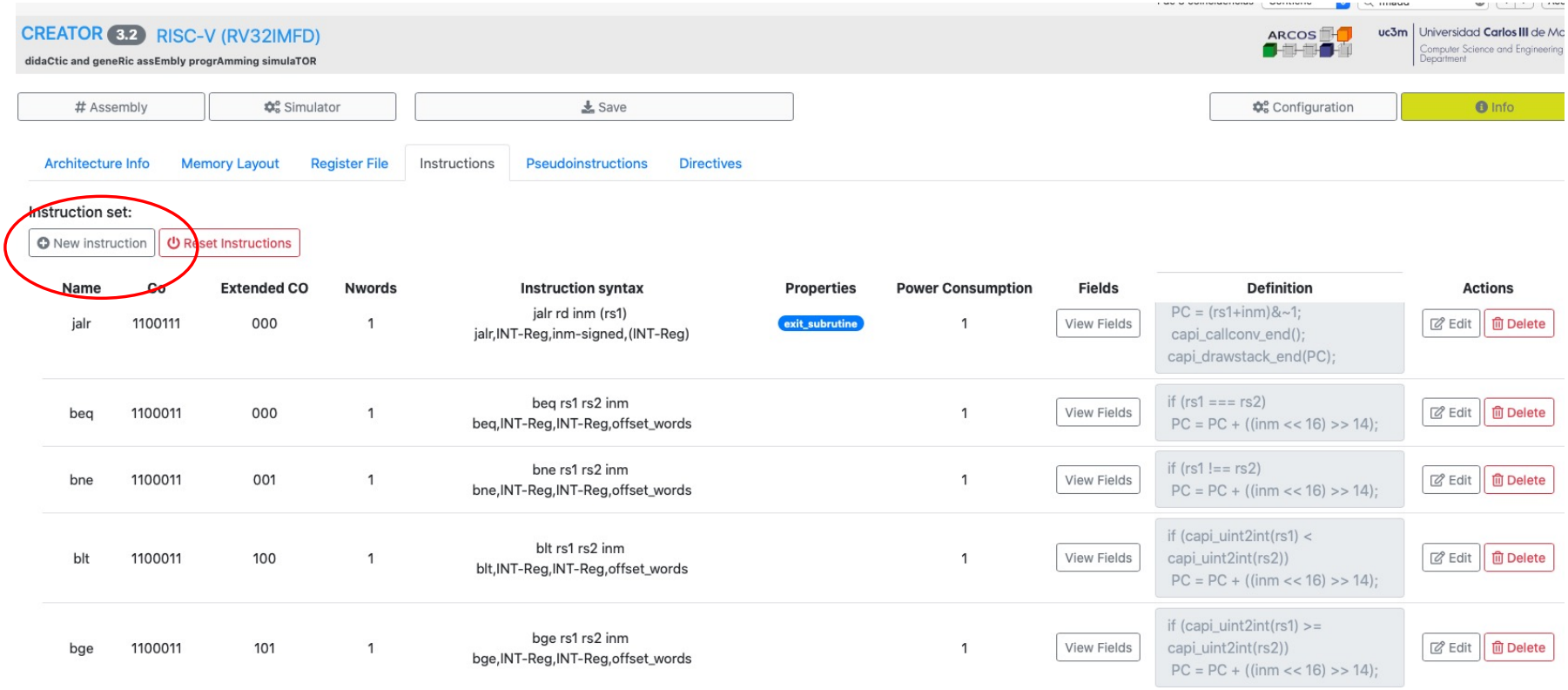
fmadd.s rd, rs1, rs2, rs3 `f[rd] = f[rs1]×f[rs2]+f[rs3]`

Floating-point Fused Multiply-Add, Single-Precision. Tipo R4, RV32F y RV64F.

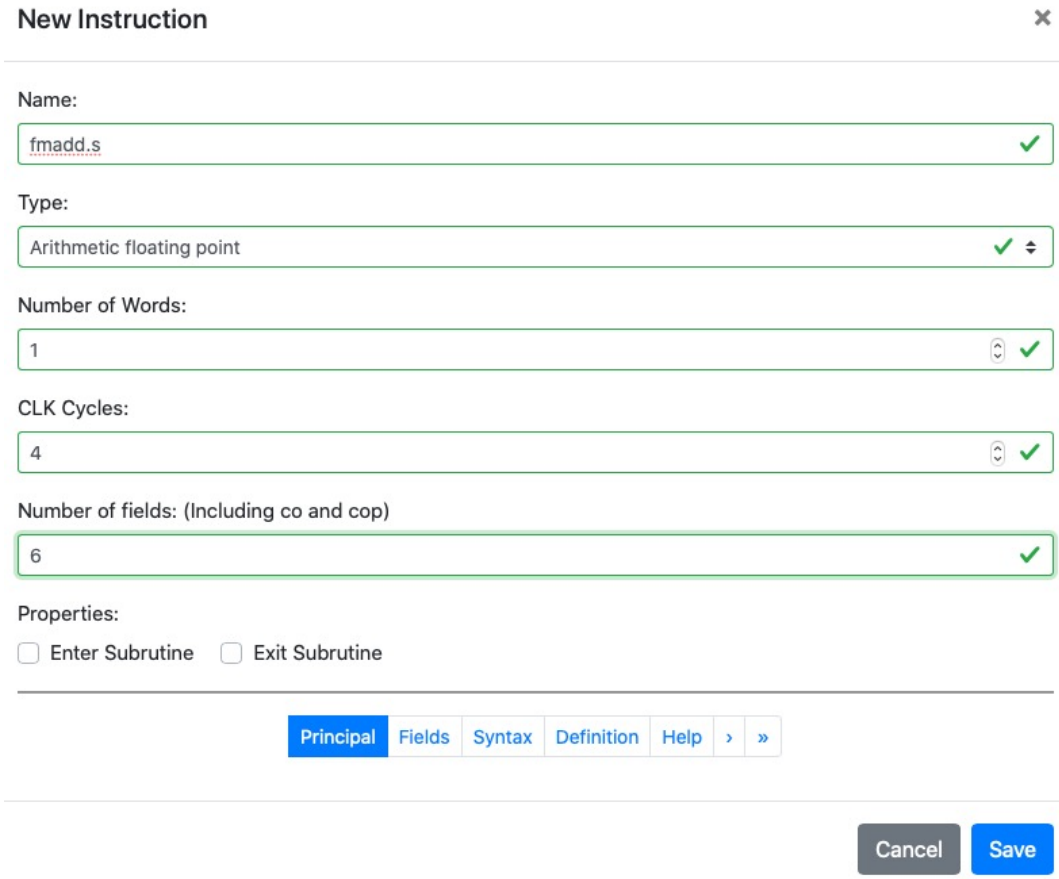
Multiplica los números de punto flotante de precisión simple en f[*rs1*] y f[*rs2*], suma el producto sin redondear al número de punto flotante de precisión simple en f[*rs3*], y escribe el resultado redondeado de precisión simple en f[*rd*].



Creación de una nueva instrucción



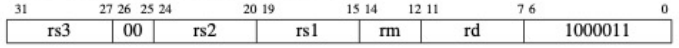
Nueva instrucción: `fmadd.s`



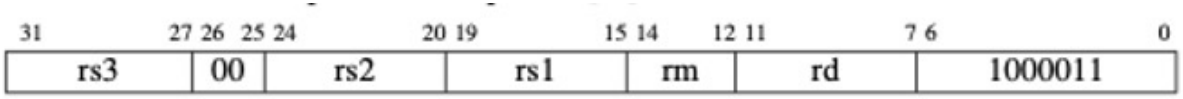
fmadd.s rd, rs1, rs2, rs3 `f[rd] = f[rs1]×f[rs2]+f[rs3]`

Floating-point Fused Multiply-Add, Single-Precision. Tipo R4, RV32F y RV64F.

Multiplica los números de punto flotante de precisión simple en f[*rs1*] y f[*rs2*], suma el producto sin redondear al número de punto flotante de precisión simple en f[*rs3*], y escribe el resultado redondeado de precisión simple en f[*rd*].



Nueva instrucción: `fmadd.s`



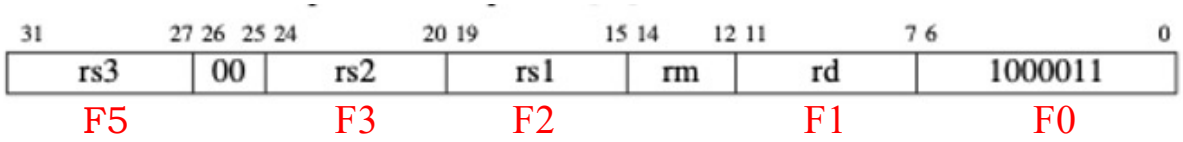
Edit fmadd.s

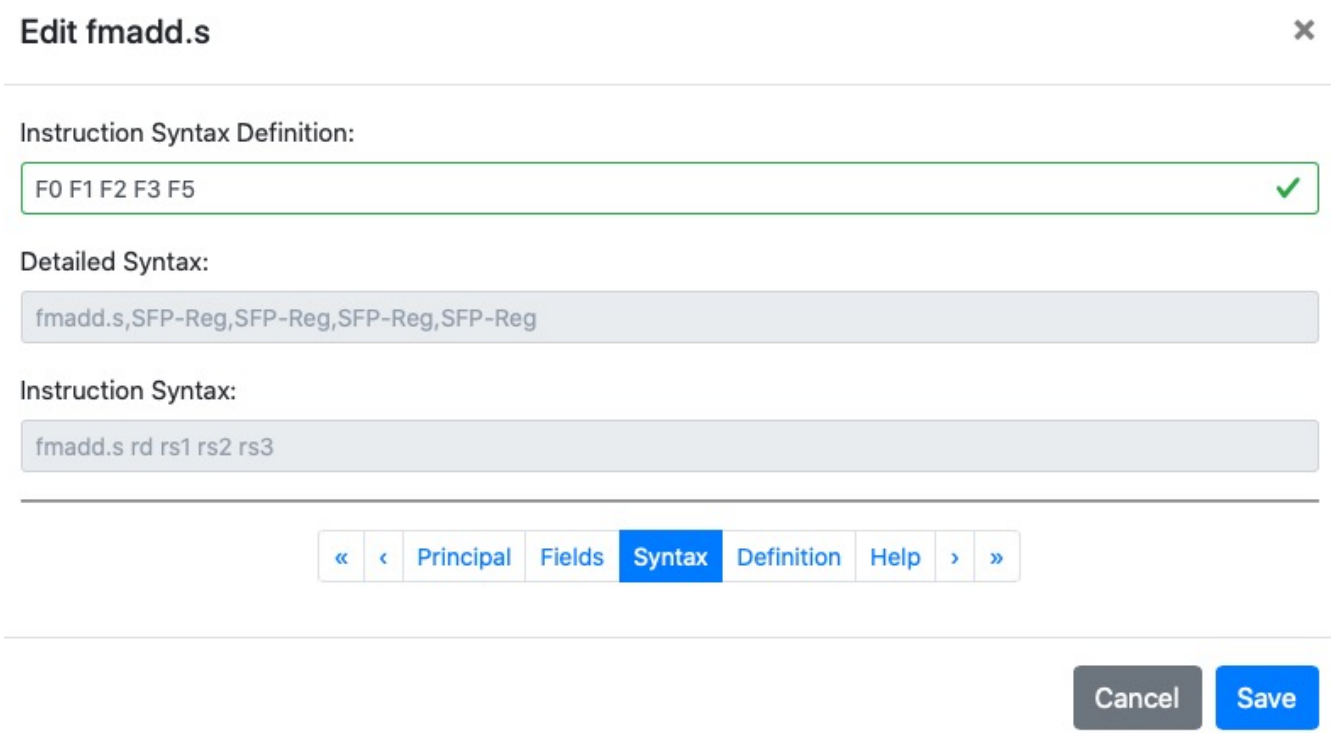
	Name:	Type	Break	Start Bit	End Bit	
Field 0	fmadd.s	co		6	0	1000011
Field 1	rd	SFP-Reg		11	7	
Field 2	rs1	SFP-Reg		19	15	
Field 3	rs2	SFP-Reg		24	20	
Field 4	co	cop		26	25	00
Field 5	rs3	SFP-Reg		31	27	

« ‹ Principal Fields Syntax Definition Help › »

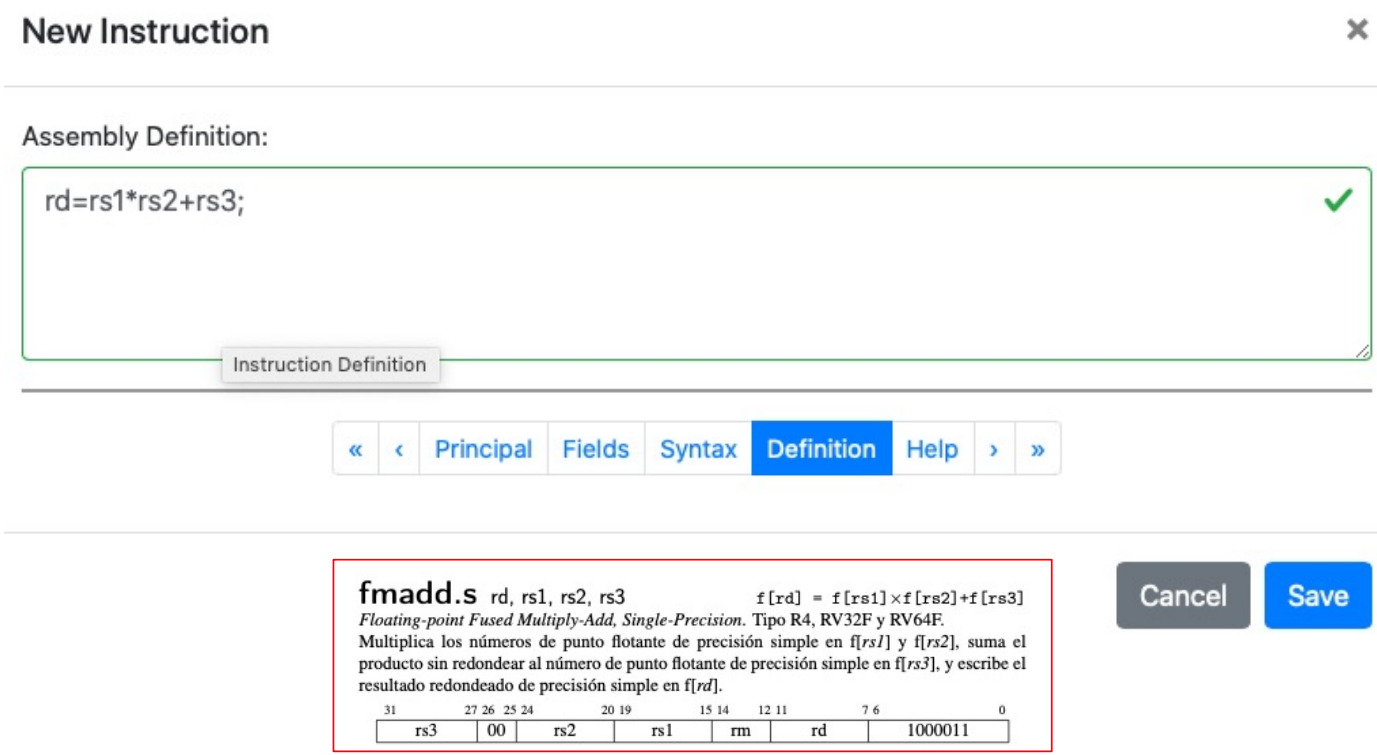
Cancel Save

Nueva instrucción: `fmadd.s`

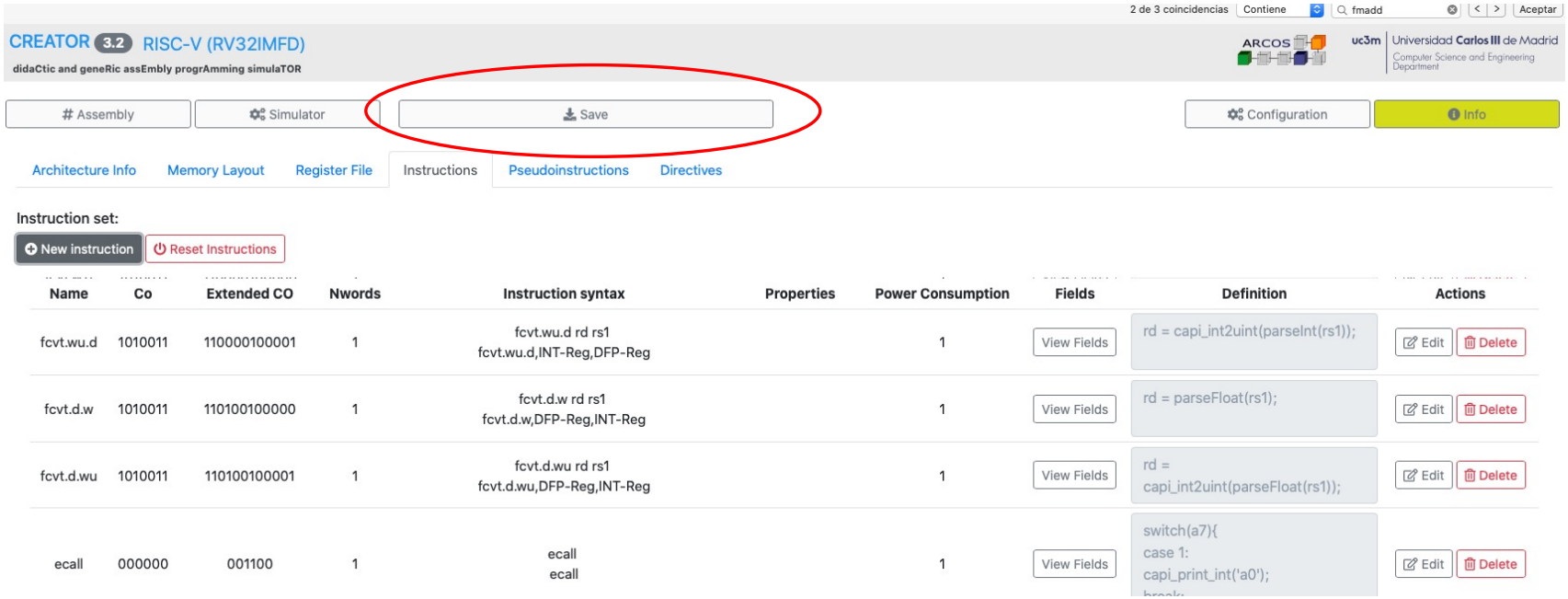




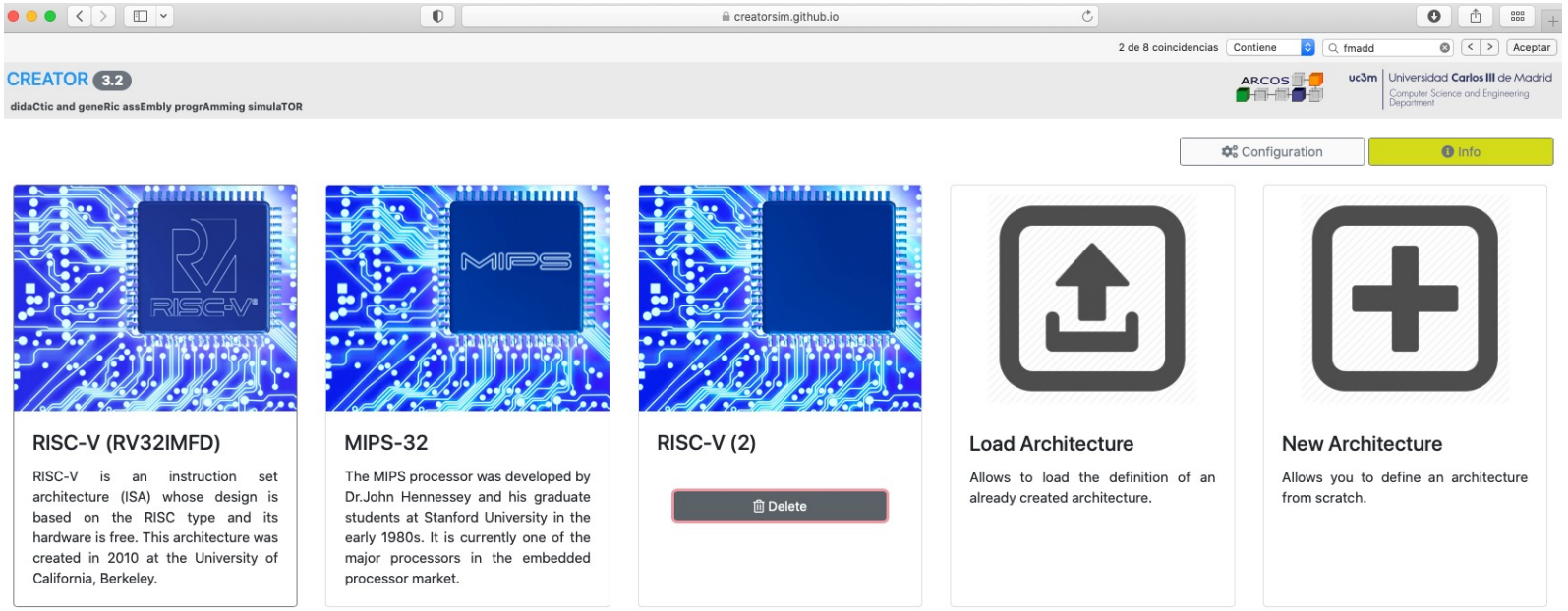
Nueva instrucción: `fmadd.s`



Creación de una nueva instrucción



Nueva arquitectura



Ejemplo

ejemplo

Assembly:

```
1  .data
2
3      a:  .float 4
4      b:  .float 5.5
5      c:  .float 2.3
6
7  .text
8
9  main:
10     li  t0, a
11     flw ft1, 0(t0)
12
13     li  t0, b
14     flw ft2, 0(t0)
15
16     li  t0, c
17     flw ft3, 0(t0)
18
19     fmadd.s fa0, ft1, ft2, ft3
20
21     li  a7, 2
22     ecall
```

API para la definición de instrucciones

- API de ayuda para definición de instrucciones
- Instrucción: `lw rd inm (rs1)`
- Definición:

```
var addr = capi_int2uint(rs1)+inm;
rd = capi_mem_read(addr, 'w', rd_name);
```

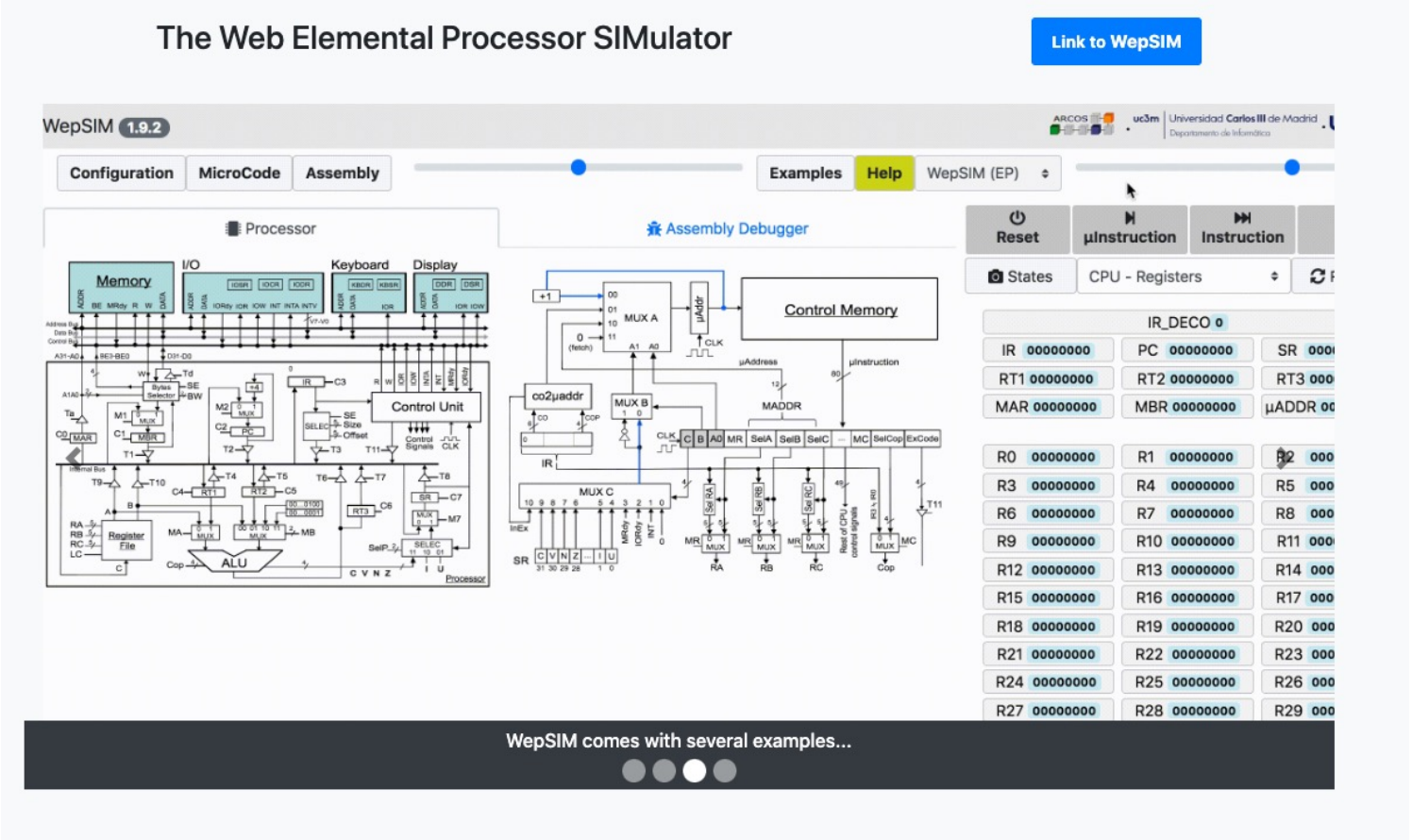
- Instrucción: `sb rd inm (rs1)`
- Definición:

```
capi_mem_write(rs1+inm, rd, 'b', rs2_name);
```

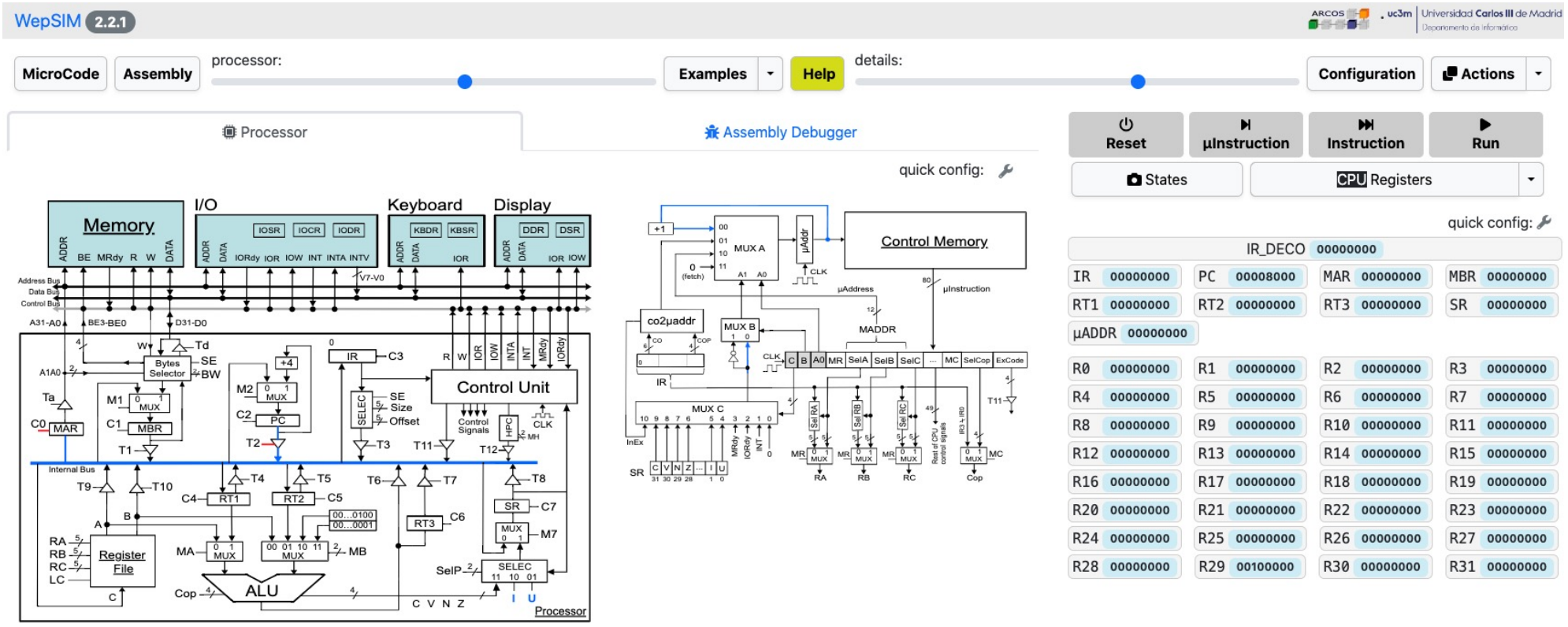
Extensiones futuras

- Simulador de caché
- Registros e instrucciones vectoriales
- Simulador de pipeline
- ….

Otro simulador: WepSim



WepSim


WepSIM 2.2.1
ARCOS
uc3m Universidad Carlos III de Madrid
Departamento de Informática
MicroCode
Assembly
processor:
Examples
Help
details:
Configuration
Actions
Processor
Assembly Debugger
quick config:
Memory
I/O
Keyboard
Display
Control Unit
Register File
ALU
Bytes Selector
Control Memory
MUX A
MUX B
MUX C
MADDR
co2µaddr
Reset
µInstruction
Instruction
Run
States
CPU Registers
quick config:
IR_DECO 00000000
IR 00000000
PC 00008000
MAR 00000000
MBR 00000000
RT1 00000000
RT2 00000000
RT3 00000000
SR 00000000
µADDR 00000000
R0 00000000
R1 00000000
R2 00000000
R3 00000000
R4 00000000
R5 00000000
R6 00000000
R7 00000000
R8 00000000
R9 00000000
R10 00000000
R11 00000000
R12 00000000
R13 00000000
R14 00000000
R15 00000000
R16 00000000
R17 00000000
R18 00000000
R19 00000000
R20 00000000
R21 00000000
R22 00000000
R23 00000000
R24 00000000
R25 00000000
R26 00000000
R27 00000000
R28 00000000
R29 00100000
R30 00000000
R31 00000000