Trabajar con Programas C y Ensamblador en RIPES

Autor: Bruno Burgos Kosmalski

Índice:

Introducción:

RIPES es un simulador que principalmente se usa para trabajar con programas escritos en ensamblador RISC-V. Sin embargo, este simulador está preparado también para ejecutar programas escritos en un lenguaje de más alto nivel como es "C".

Para poder trabajar con este tipo de programas, en esta plataforma tendremos principalmente dos opciones: trabajar a partir del propio interfaz del simulador, siendo esta la opción más básica y cómoda, aunque también la que cuenta con menos libertades; o montar el ejecutable del programa fuera y trabajar con este en el propio simulador.

Para la realización de este documento se ha trabajado con las configuraciones de RIPES para RV32I [1], lo que también implica que el compilador que se ha usado es el especializado para este mismo caso (riscv32-unknown-elf). Nótese que de trabajar con otras configuraciones u otro compilador es posible que se necesite hacer cambios en algunas de las sentencias que se emplean en los ejemplos de este documento.

Configuración y requisitos previos:

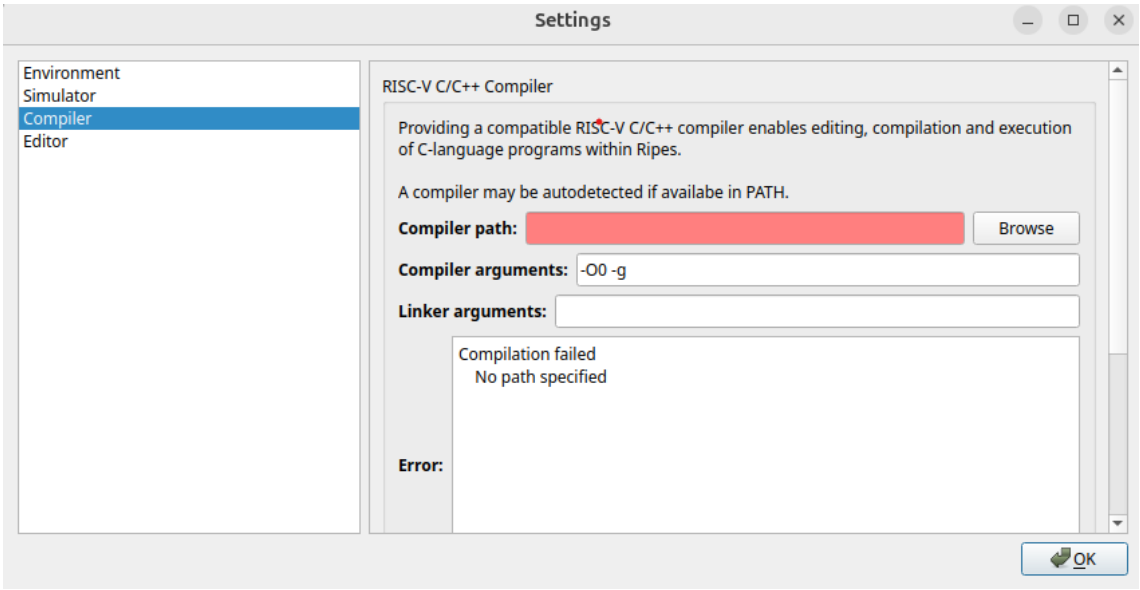
Para trabajar con programas "C" en RIPES, a diferencia de los programas ensamblador RISC-V, necesitaremos instalar a parte el compilador específico que nos permitirá crear los archivos ejecutables. Esto se puede hacer de diferentes formas: se puede hacer a través del repositorio oficial [2] donde encontramos el código fuente sobre el que crearemos todos los ejecutables, opción que no recomiendo por lo compleja y pesada que se puede hacer; se puede instalar a través de apt, siempre que contemos con esta opción; o se puede instalar una versión precompilada y añadirla a la ruta de ejecución. Esta última opción es la que se ha usado para la realización de este documento, donde los ejecutables se han sacado del repositorio no oficial con enlace: https://github.com/stnolting/riscv-gcc-prebuilt.

Una vez instalado el compilador que vamos a usar necesitaremos configurar RIPES para que este lo use, cabe resaltar que necesitaremos para esto la ruta absoluta donde se encuentra el ejecutable.

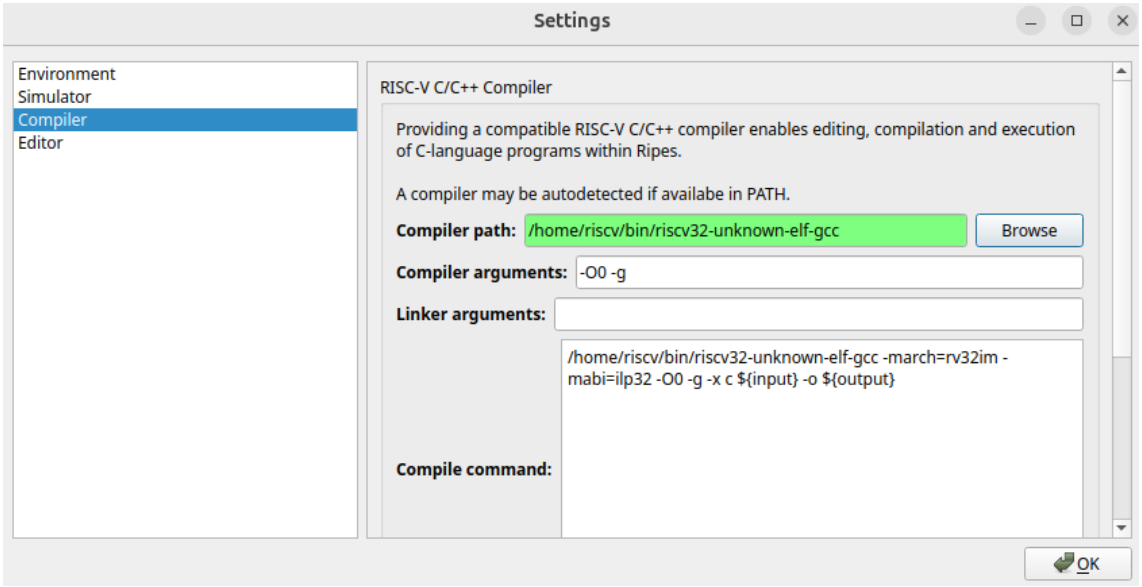
Pasos a seguir:

Primero tendremos que entrar en el apartado "Settings" dentro del apartado "Edit" en la esquina superior izquierda.

Una vez abierta la pestaña tendremos que dirigirnos al apartado "Compiler", más concretamente tendremos que ir a donde pone "Compiler path" y añadir la ruta absoluta del compilador.



Si todo se ha ido bien el apartado se iluminará de color verde indicando que ha detectado el ejecutable y está preparado para usarlo.



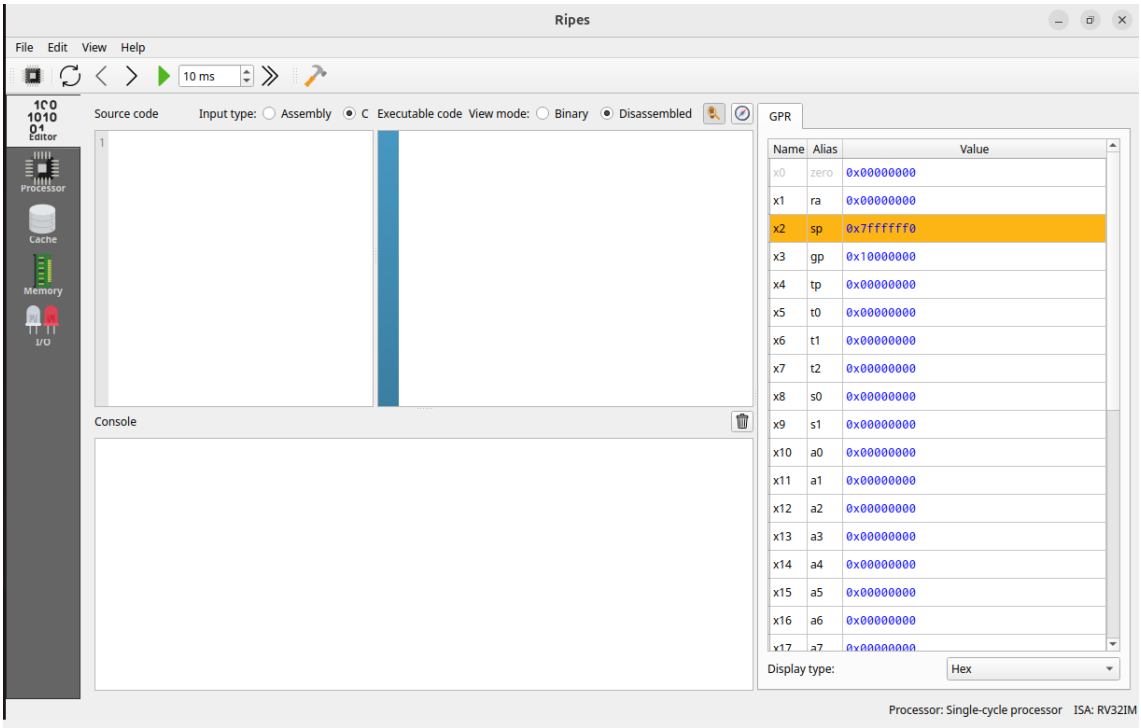
Una vez hecho esto, ya lo tendremos todo listo para empezar a trabajar con programas escritos en “C”.

Trabajar con la Interfaz de RIPES:

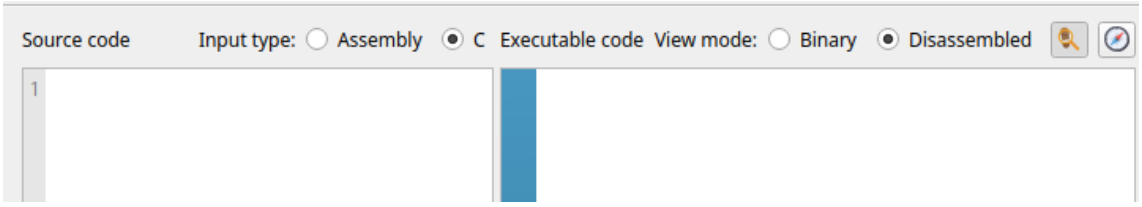
Si queremos trabajar con programas escritos en “C” usando la propia interfaz del simulador RIPES, podremos hacerlo teniendo en cuenta una serie de restricciones: los programas que hagamos tendrán que estar escritos en un único fichero; no podremos trabajar varios lenguajes al mismo tiempo, sino que todo lo que escribamos tendrá que estar escrito en “C”; si queremos tener programas más complejos que requieran de opciones adicionales al compilador, tendremos que especificar las mismas en el apartado de configuración.

Una vez dicho esto, se mostrará un ejemplo sencillo de un programa escrito en “C”, y los pasos necesarios para ejecutarlo usando este simulador.

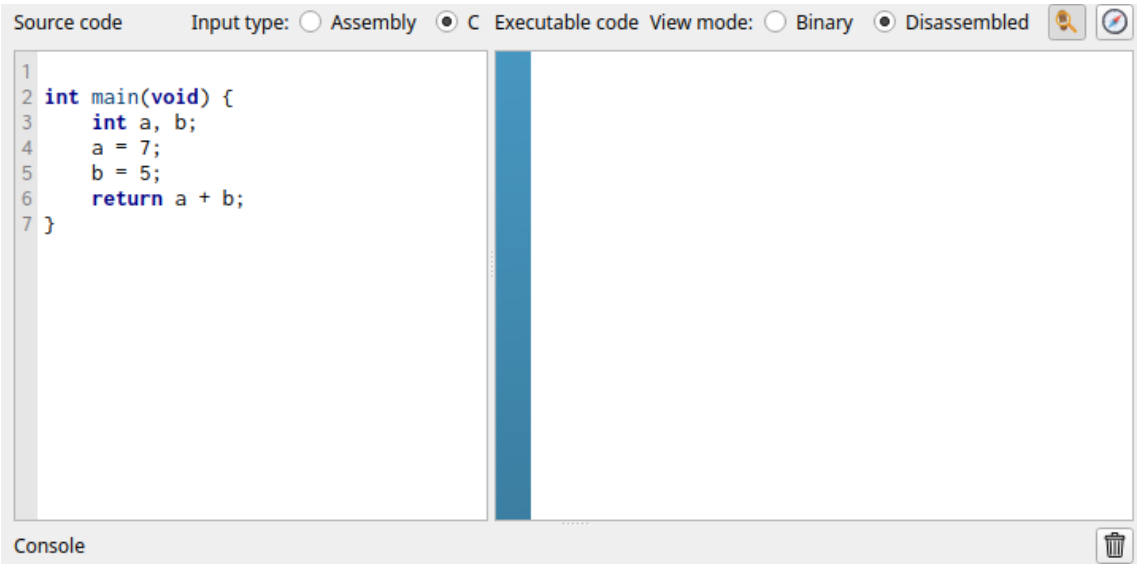
Empezamos abriendo el apartado “Editor” donde vamos a trabajar a lo largo de todo este proceso.



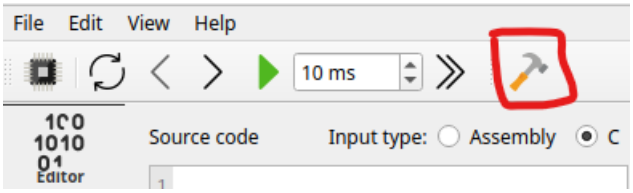
Justo encima de la consola, en la pestaña izquierda, es donde tendremos el editor en el que podremos escribir el código “C”, importante seleccionar la opción “C” justo encima del editor.



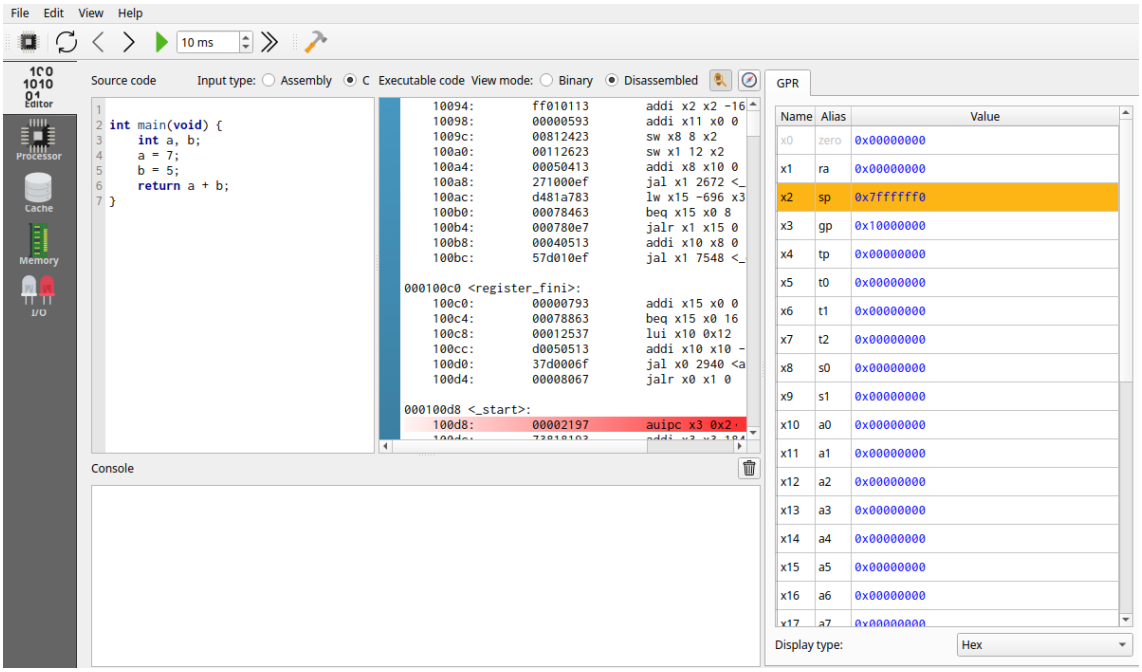
Ahora escribimos un programa sencillo en “C”. Para este ejemplo he elegido un programa principal que suma dos números enteros y retorna el valor resultante.

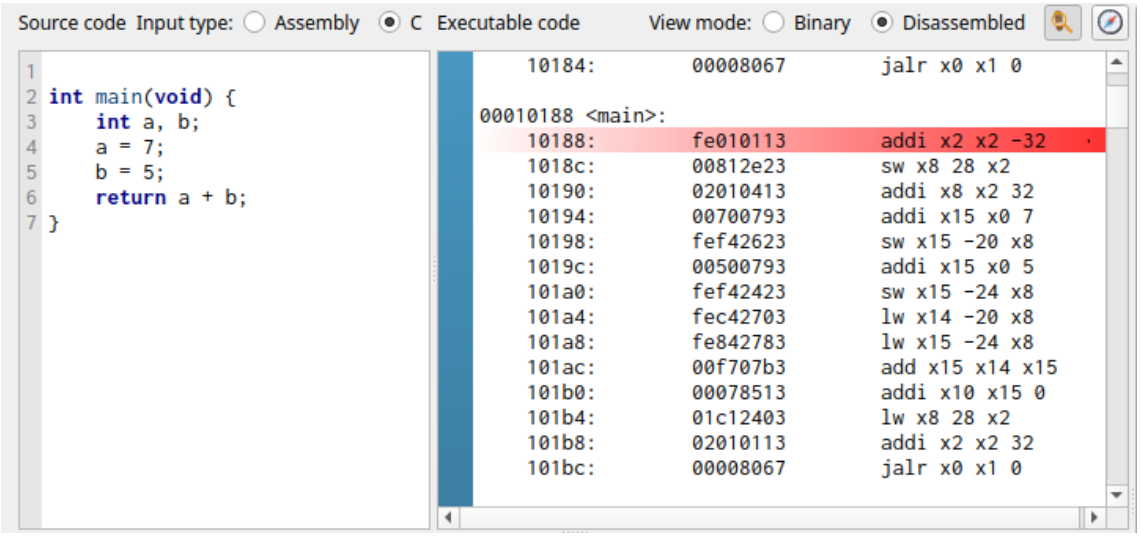


Una vez hemos terminado el programa, construimos el ejecutable y lo cargamos. Esto se hace pulsando el botón con el símbolo de martillo, para simbolizar la construcción.

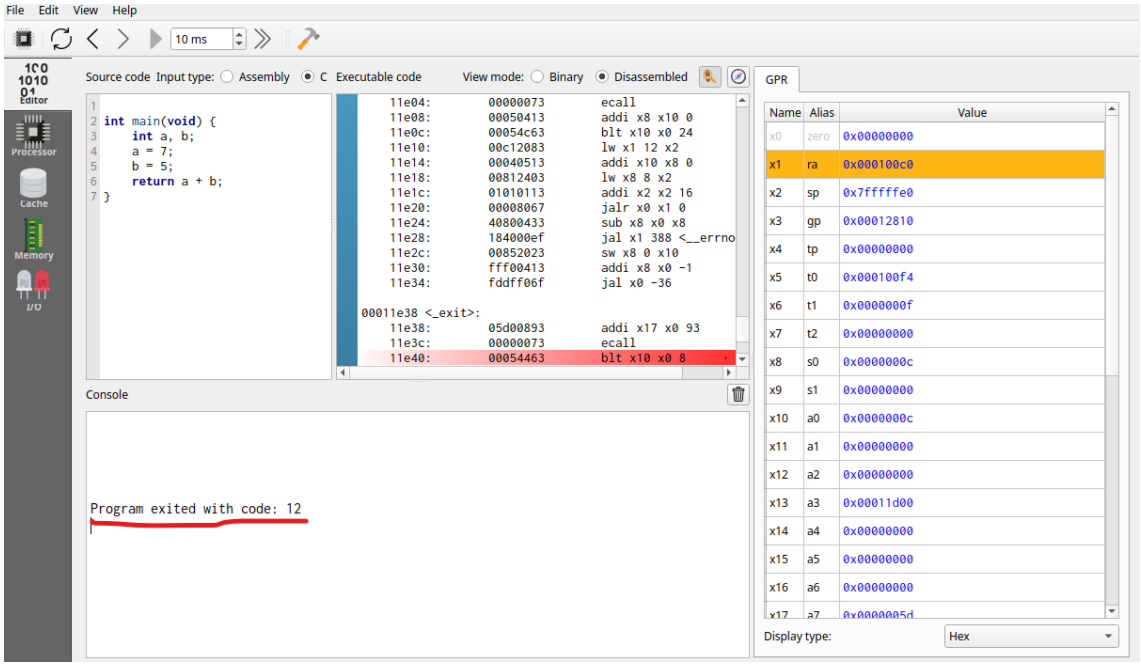


Si todo ha progresado de forma correcta se debería haber generado el código desensamblado a la derecha del editor, que es donde vamos a ver la ejecución de las instrucciones una a una, aunque esto no se refleje de forma directa sobre el código “C”.

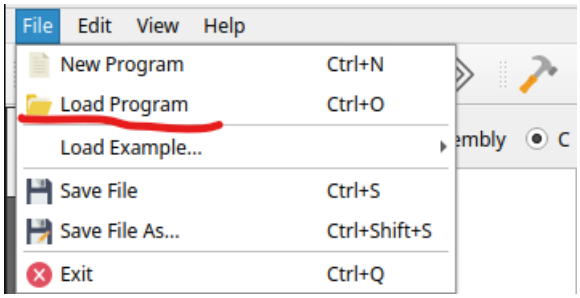




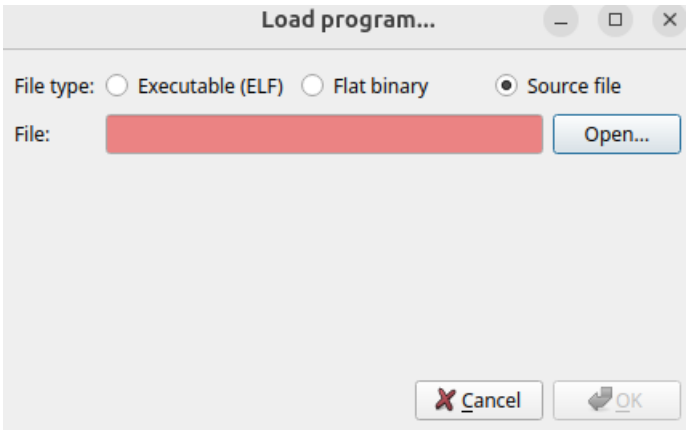
Ahora podemos ir ejecutando instrucción a instrucción, añadiendo breackpoints y viendo cómo cambia el valor de los registros, o podemos ejecutar todo el código de un tirón. En este ejemplo, una vez finalizada la ejecución del programa debería aparecer por pantalla el valor de salida, que en este caso debería ser 12.

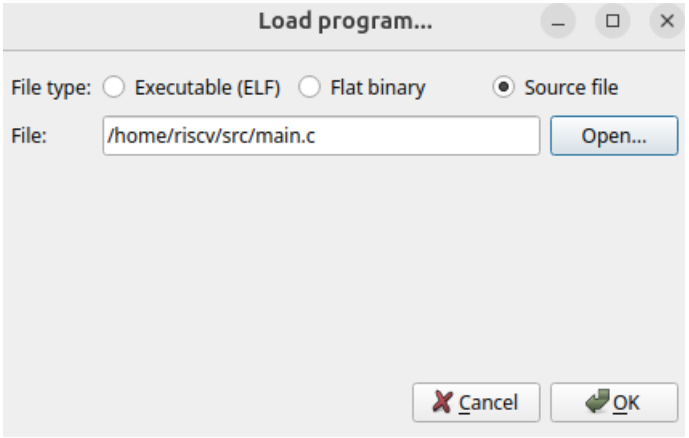


Por otra parte, si ya se cuenta con un programa escrito en “C”, este se puede cargar desde la ruta en la que se encuentre directamente al simulador. Para esto se tiene que abrir el apartado “File”, más concretamente el apartado “Load Program”.



Dentro de este apartado seleccionamos la opción “Source File” y le damos la ruta absoluta del código que queremos cargar en el simulador.





Trabajar a partir de Ejecutables:

Si queremos trabajar con programas más complejos, formados de varios archivos, o incluso, si queremos mezclar varios lenguajes como puede ser ensamblador RISC-V con "C", vamos a necesitar montar el ejecutable aparte, y cargarlo sobre RIPES para ver el comportamiento a bajo nivel del mismo. Para esto se van a mostrar a continuación algunos ejemplos con "C" y ensamblador.

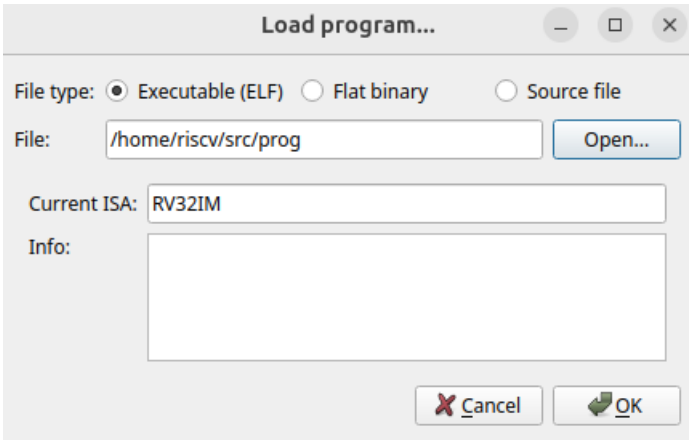
Empezando por mostrar como se haría el caso anterior montando el ejecutable desde fuera, lo primero que tenemos que hacer es compilarlo. Para esto empleamos la sentencia "riscv32-unknown-elf-gcc main.c -g -c -o main.o" que nos genera el programa objeto. A partir de este podemos ver el ensamblador correspondiente a la función principal empleando la sentencia "riscv32-unknown-elf-objdump -S main.o".

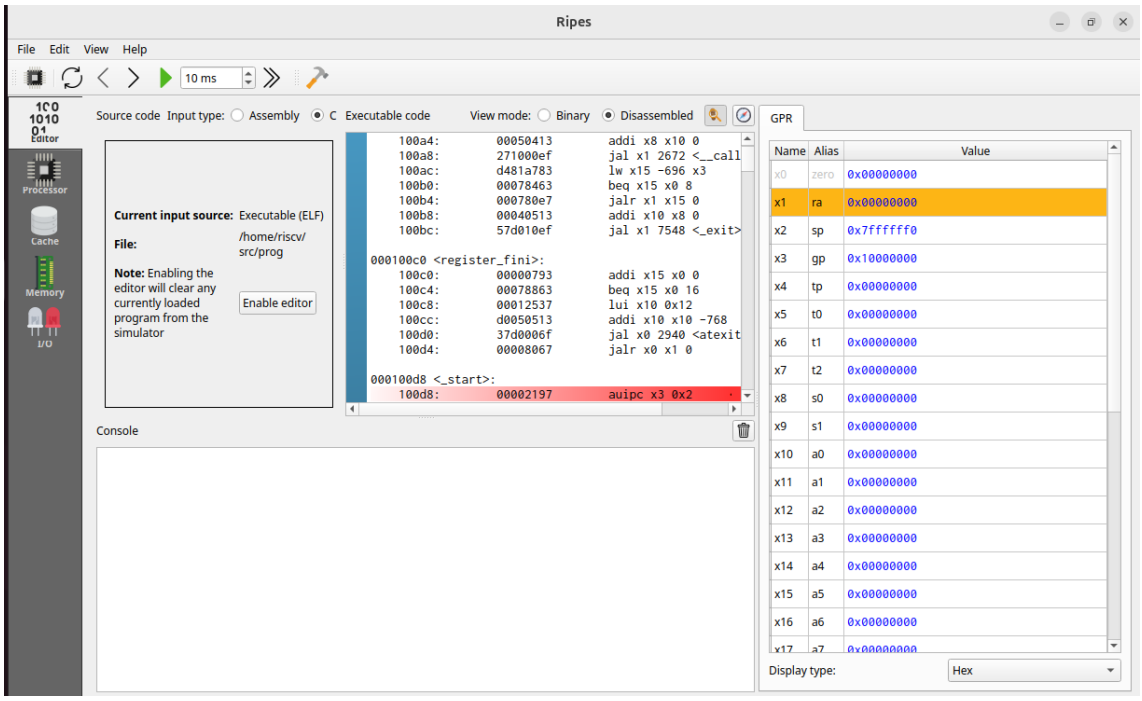
```
root@MVUbuntu:/home/riscv/src# riscv32-unknown-elf-objdump -S main.o

main.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <main>:

int main(void) {
   0:   fe010113                add     sp,sp,-32
   4:   00812e23                sw      s0,28(sp)
   8:   02010413                add     s0,sp,32
   int a, b;
   a = 7;
   c:   00700793                li      a5,7
  10:   fef42623                sw      a5,-20(s0)
   b = 5;
  14:   00500793                li      a5,5
  18:   fef42423                sw      a5,-24(s0)
   return a + b;
  1c:   fec42703                lw      a4,-20(s0)
  20:   fe842783                lw      a5,-24(s0)
  24:   00f707b3                add     a5,a4,a5
}
  28:   00078513                mv      a0,a5
  2c:   01c12403                lw      s0,28(sp)
  30:   02010113                add     sp,sp,32
  34:   00008067                ret
root@MVUbuntu:/home/riscv/src#
```

Una vez que tenemos los objetos que necesitamos solo haría falta construir el ejecutable a partir de estos, como si se tratase de un programa "C" estándar, para este ejemplo usamos la sentencia "riscv32-unknown-elf-gcc main.o -o prog". Si usamos la sentencia anterior a esta podemos ver también el código ensamblador, aunque a diferencia del objeto, el programa tendrá también el resto de las funciones de librería, lo que puede dificultar el encontrar las sentencias que realmente nos interesan.

```
root@MVUbuntu:/home/riscv/src# riscv32-unknown-elf-objdump -S prog

prog:     file format elf32-littleriscv


Disassembly of section .text:

00010094 <exit>:
   10094:       ff010113                add     sp,sp,-16
   10098:       00000593                li      a1,0
   1009c:       00812423                sw      s0,8(sp)
   100a0:       00112623                sw      ra,12(sp)
   100a4:       00050413                mv      s0,a0
   100a8:       271000ef                jal     10b18 <__call_exitprocs>
   100ac:       d481a783                lw      a5,-696(gp) # 12558 <__stdio_exit_handler>
   100b0:       00078463                beqz    a5,100b8 <exit+0x24>
   100b4:       000780e7                jalr    a5
   100b8:       00040513                mv      a0,s0
   100bc:       57d010ef                jal     11e38 <_exit>

000100c0 <register_fini>:
   100c0:       00000793                li      a5,0
   100c4:       00078863                beqz    a5,100d4 <register_fini+0x14>
   100c8:       00012537                lui     a0,0x12
   100cc:       d0050513                add     a0,a0,-768 # 11d00 <__libc_fini_array>
   100d0:       37d0006f                j       10c4c <atexit>
   100d4:       00008067                ret

000100d8 <_start>:
   100d8:       00002197                auipc   gp,0x2
   100dc:       73818193                add     gp,gp,1848 # 12810 <__global_pointer$>
```

Una vez que tenemos el ejecutable ya montado solo hace falta cargarlo en RIPES. El proceso es similar al de cargar un código “C” o ensamblador a partir de un fichero ya existente. Para ello nos vamos a la interfaz “File”, “Load Program” y seleccionamos la opción “Executable (Elf) ”, le damos la ruta absoluta del programa ejecutable que queremos montar, y si todo ha ido correctamente, entonces ya podríamos simular el programa desde RIPES. Se seguirán estos mismos pasos para los siguientes ejemplos a la hora de cargar los programas en RIPES.





Pasando al siguiente nivel, ahora vamos a hacer un programa sencillo en “C” que integre también funciones escritas en ensamblador. Para esto, le damos a la función un nombre que vayamos a usar tanto en el código “C” como en el código ensamblador, y la declaramos como extern. Hay que tener en cuenta como realiza el compilador el paso de parámetros para las funciones, en el caso de RISC-V el paso de parámetros siempre que se puede, se hace mediante registros. Tomando el ejemplo anterior, si declaramos una función suma, veremos que para pasar los dos enteros se usan los registros a0 y a1, mientras que para el valor de retorno se usa el valor a1.

```c
extern int sum(int a, int b);

int main(void) {
   int a, b;
   a = 7;
   b = 5;
   return sum(a,b);
}
```

Por otra parte, para el código ensamblador tenemos que declarar como globales todas aquellas funciones que vamos a usar en el programa, esto se hace con la directiva del compilador “.globl”, luego solo tenemos que escribir el código asociado a la función.

```
.globl sum
.text
sum:
add a0, a0, a1
ret
```

Una vez que tenemos todos los códigos que necesitamos para montar el programa, tendremos que crear el programa a partir de los mismos, o hacerlo directamente. Una vez

hecho solo tenemos que seguir los mismos pasos del ejemplo anterior, cargando el programa en RIPES para poder ejecutarlo.

```
root@MVUbuntu:/home/riscv/src# riscv32-unknown-elf-gcc main.o sum.o -o prog
root@MVUbuntu:/home/riscv/src# riscv32-unknown-elf-objdump -S prog

prog:     file format elf32-littleriscv


Disassembly of section .text:

00010094 <exit>:
   10094:       ff010113                add     sp,sp,-16
   10098:       00000593                li      a1,0
   1009c:       00812423                sw      s0,8(sp)
   100a0:       00112623                sw      ra,12(sp)
   100a4:       00050413                mv      s0,a0
   100a8:       285000ef                jal     10b2c <__call_exitprocs>
   100ac:       d481a783                lw      a5,-696(gp) # 12558 <__stdio_exit_handler>
   100b0:       00078463                beqz    a5,100b8 <exit+0x24>
   100b4:       000780e7                jalr    a5
   100b8:       00040513                mv      a0,s0
   100bc:       591010ef                jal     11e4c <_exit>

000100c0 <register_fini>:
   100c0:       00000793                li      a5,0
   100c4:       00078863                beqz    a5,100d4 <register_fini+0x14>
   100c8:       00012537                lui     a0,0x12
   100cc:       d1450513                add     a0,a0,-748 # 11d14 <__libc_fini_array>
   100d0:       3910006f                j       10c60 <atexit>
   100d4:       00008067                ret
```

Ya para finalizar con los ejemplos, que ocurriría si en vez de declarar una subrutina hoja en ensamblador, quisiésemos traducir una función que llama a otras funciones de “C” a ensamblador, pero solo esta función y no las que llama. Pues podemos hacer esto si dentro del código ensamblador declaramos estas funciones adicionales como externas, lo que se hace con la directiva del compilador “.extern”. Siempre, claro está, teniendo en cuenta que el paso de parámetros que usemos coincida con el que usa el compilador por defecto.

Para este ejemplo, vamos a crear un programa sencillo en “C” que realiza la suma del factorial de dos números enteros y la retorna como valor de salida. Y luego vamos a traducir la función que los suma, pero no la función que hace la operación factorial de un número entero, sino que la vamos a dejar tal cual está en el programa “C”.

```c
int fact(int a) {
    int b = a;
    int res = 1;
    while(b > 1) {
         res *= b;
         b--;
    }
    return res;
}

int sum_fact(int a, int b) {
    return fact(a) + fact(b);
}

int main(void) {
    int a, b;
    a = 7;
    b = 5;
    return sum_fact(a,b);
}
```

Cambios para pasar la función sum_fact a ensamblador.

```c
extern int sum_fact(int a, int b);

int fact(int a) {
    int b = a;
    int res = 1;
    while(b > 1) {
         res *= b;
         b--;
    }
    return res;
}

int main(void) {
    int a, b;
    a = 7;
    b = 5;
    return sum_fact(a,b);
}
```

Código ensamblador para la función sum_fact.

```
.extern fact
.globl sum_fact
.text
sum_fact:
   addi sp, sp, -8
   sw ra, 4(sp)
   sw a1, 0(sp)
   jal ra, fact
   or a1, a0, a0
   lw a0, 0(sp)
   sw a1, 0(sp)
   jal ra, fact
   lw a1, 0(sp)
   add a0, a0, a1
   lw ra, 4(sp)
   addi sp, sp, 8
   ret
```

Si ejecutamos este programa en RIPES, podemos ver que el valor de salida coincide con la operación 5! + 7!, que es la operación que hemos programado en el código anterior por lo que hemos podido comprobar que se puede mezclar código ensamblador con código “C” sin mayor complicación.

Resumen:

En los apartados anteriores hemos visto cómo trabajar con códigos escritos en lenguaje “C”, donde hemos podido ver que hay dos formas principales de trabajar con ello.

Podemos trabajar con la interfaz que nos ofrece el simulador RIPES cargando el código "C" en su editor. Esta forma de trabajar es bastante cómoda, ideal para trabajar con programas sencillos montados en un único archivo. Sin embargo, también ofrece pocas libertades, siendo que para los programas más complejos o que se montan sobre varios ficheros no nos es suficiente.

Por otra parte, si buscamos trabajar con códigos más complejos: que mezclen ensamblador con lenguaje “C”, que estén repartidos sobre varios ficheros, etc. RIPES nos ofrece otra opción, como es crear el ejecutable por fuera y cargarlo en el simulador para que le ejecución se haga desde dentro. Esta opción puede no ser tan cómoda como la anterior, pero si buscamos salir de las limitaciones que nos impone, o buscamos una solución especializada es la única opción de la que disponemos.

Para hacer más sencillo el trabajo anterior, añado a continuación los mandatos que he empleado para generar los ejecutables en los ejemplos, con una breve descripción para cada una de estos.

Para generar el código objeto de un programa en cuestión, esté escrito en “C” o en ensamblador usamos el compilador de riscv.

> “riscv32-unknown-elf-gcc cod.c -g -c -o cod.o”
>
> “riscv32-unknown-elf-gcc cod.s -g -c -o cod.o”

La opción “-g” añade las opciones para depuración, esenciales si queremos visualizar el código “C” al que corresponden las líneas de código máquina en el visualizador (objdump) por ejemplo.

La opción “-s” nos permite generar el código ensamblador a partir de un código “C”, por ejemplo:

> “riscv32-unknown-elf-gcc cod.c -s cod.s”

Donde cod.c es el código “C” y cod.s es el código ensamblador correspondiente al mismo.

Una vez tenemos todos los objetos que necesitamos para montar el programa, podemos hacerlo también usando el compilador riscv, al igual que lo haríamos con un compilador de “C” estándar como es el gcc.

> “riscv32-unknown-elf-gcc main.o obj1.o obj2.o ... -o prog”

También lo podemos hacer directamente sin tener que generar los objetos, al igual que lo haríamos con un compilador de “C” estándar.

> “riscv32-unknown-elf-gcc main.c obj1.c obj2.c ... -o prog”

Una vez hemos generado los objetos, o el programa ejecutable, como estos ya están en código máquina que no podemos entender, si queremos visualizar su contenido podemos usar la herramienta objdump con la opción -S.

“riscv32-unknown-elf-objdump -S prog”

Que nos mostrará las instrucciones máquina que conforman el programa que queremos visualizar.

Para resumir, suponiendo que tuviésemos un programa principal escrito en “C” (main.c), con dos códigos adicionales, uno en ensamblador (obj1.s) y otro escrito en “C” (obj2.s). Si quisiéramos crear el ejecutable final y visualizar su contenido, tendríamos que ejecutar las siguientes sentencias:

“riscv32-unknown-elf-gcc main.c -c -o main.o”

“riscv32-unknown-elf-gcc obj1.s -c -o obj1.o”

“riscv32-unknown-elf-gcc obj2.c -c -o obj2.o”

“riscv32-unknown-elf-gcc main.o obj1.o obj2.o -o prog”

También se puede usar: “riscv32-unknown-elf-gcc main.c obj1.s obj2.c -o prog”

“riscv32-unknown-elf-objdump -S prog”

Referencias:

[1] https://drive.google.com/file/d/1uviu1nH-tScFfgrovvFCrj7Omv8tFtkp/view

[2] https://github.com/riscv-collab/riscv-gnu-toolchain

[3] .....

[4] .....