HowTo: Debug AArch64 GDB

Instalación

0-TENER ACTUALIZADOS LOS REPOSITORIOS

```
$ sudo apt update
```

1- SETTING UP AARCH64 TOOLCHAIN

```
$ sudo apt install gcc-aarch64-linux-gnu
```

2- SETTING UP QEMU ARM (incluye aarch64)

```
$ sudo apt install qemu-system-arm
```

3- FETCH AND BUILD AARCH64 GDB

```
$ sudo apt install gdb-multiarch
```

4- CONFIGURAR GDB PARA QUE HAGA LAS COSAS MÁS AMIGABLES

```
$ wget -P ~ git.io/.gdbinit
```

Esto crea un archivo llamado .gdbinit en el directorio personal que configura el GDB para funcionar como un Dashboard.

Ensamblado

1 - Obtener los archivos del moodle (link), descomprimirlos y situarse en la carpeta mediante la terminal.

```
$ cd ./P6Ej7/ej7
```

2 - Escribir el programa a simular en el template main.s

3 - Compilar utilizando el Makefile

```
$ make
```

Inicio del emulador

1 - Iniciar el emulador del microprocesador ARM
Utilizando el make, se puede ejecutar el QEMU de la siguiente manera:

```
$ make runQEMU
```

En caso contrario, se debe ejecutar con el siguiente comando (verificar que se haya copiado y pegado el comando completo):

```
$ qemu-system-aarch64 -s -S -machine virt -cpu cortex-a53 -machine type=virt -nographic -smp 1 -m 64 -kernel kernel.img
```

Al ejecutar este comando, la terminal queda ejecutandolo. Para continuar con la ejecución del dashboard, se debe abrir una nueva terminal.
Nota: Cada vez que se compile, se deberá reiniciar el emulador, para cerrarlo se debe presionar ctrl a + x

Inicio del debugger

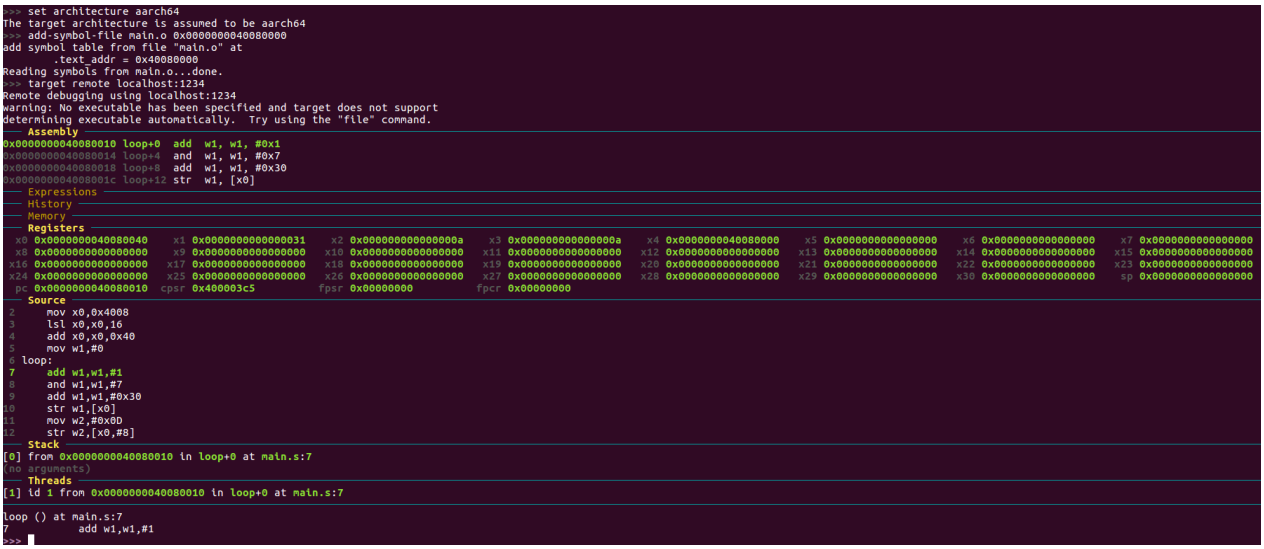
1 - Iniciar debugger GDB (Este comando se debe ejecutar en una terminal diferente a la del emulador)
Utilizando el make, se puede iniciar el GDB de la siguiente manera:
$ make runGDB
En caso contrario, se debe ejecutar con el siguiente comando:

```
$ gdb-multiarch -ex "set architecture aarch64" \
-ex "target remote localhost:1234"
```

2 - Importar al GDB los símbolos de debug en la dirección de memoria donde se encuentra el programa

```
>>> add-symbol-file main.o 0x0000000040080000
```

3 - Como resultado se debería obtener algo similar a lo mostrado en la siguiente imagen:



Nota: Para cerrarlo se debe presionar ctrl d.

Configuración del Dashboard

El Dashboard es un interfaz visual modular que se utiliza para mostrar de forma más amigable la información relevante para realizar el debugging. Algunas de las secciones más importantes son:

- **Assembly**: Muestra la próxima instrucción a ejecutarse (en verde) y algunas de las siguientes.
- **Memory**: Contenido de un segmento especificado de la memoria.

- **Registers**: Valor actual de todos los registros, aquellos que se modifican se cambian de color a verde.
- **Source**: Código fuente y la instrucción a ejecutarse resaltada en verde.

Comandos útiles de configuración del dashboard

- Mostrar un segmento de memoria:
Considerando que *dir* es la dirección base de la memoria a partir de donde se desea ver el contenido y *n_byte* la cantidad de bytes que se mostrarán a partir de dicha dirección, se debe utilizar el siguiente comando:

```
>>> dashboard memory watch dir  n_byte
```

Ejemplo:

```
>>> dashboard memory watch 0x0000000040080000 128
```

- Eliminar del dashboard los segmentos de memoria que se están mostrando:

```
>>> dashboard memory clear
```

- Mostrar en la sección "Assembly" el opcode de una instrucción que se está por ejecutar:

```
>>> dashboard assembly -style opcodes 1
```

Ejecución paso a paso con GDB

El comando que se utiliza para ejecutar una única instrucción de assembly es:

```
>>> stepi
```

Si se desea ejecutar un bloque de *n* instrucciones:

```
>>> stepi n
```

Otra forma de ejecutar más de una instrucción es utilizando *breakpoints*, esto se puede hacer de dos formas, indicando el número de instrucción (*n_instr*) o mediante una etiqueta agregada en el código fuente (*etiqueta*)

```
>>> break n_instr
>>> break etiqueta
```

Luego de indicar los breakpoints, mediante el comando *continue* se ejecuta el programa hasta encontrar el primer *breakpoint*.

```
>>> continue
```

Para ver todos los *breakpoints* existentes, se utiliza el comando:

```
>>> info breakpoints
```

Para eliminar todos *breakpoints* se utiliza el comando *delete*:

```
>>> delete breakpoints
```