RISC-V GDB

TUTORIAL

Developed by: SHAKTI Development Team @ iitm ’20

https://shakti.org.in

0.1 Proprietary Notice

0.2 Release Information

Version	Date	Changes
0.1		**Initial release**

RISC-V GDB Tutorial

Table of contents

1.1 Introduction

GDB can be used to examine the contents of memory and registers while executing a program, thereby enabling one to debug a program. This is done by using gdb commands to

- Start executing program
- Stop or pause execution at specified conditions
- Examine the contents of registry or memory locations
- Alter contents or registry or memory location

This tutorial briefly explains about using GDB to debug a program and about the GDB commands.

1.2 Steps to Use GDB

1. Compile the program.
2. Start GDB
   For RISC-V 64 bit use
   `riscv64-unknown-elf-gdb`
   For RISC-V 32 bit use
   `riscv32-unknown-elf-gdb`
3. Specify the program to be debugged
   `file <program to be debugged >`
4. Connect to remote target
   `target remote <address:port number>`
5. Load the file to memory
   `load`
6. Set breakpoints
   `break <filename>:<line number>`
7. Execute the program
   `continue`
   Or
   `jump <address>`
8. Step through the program
   `stepi`
9. Exit GDB
   `quit or q`

2. Commands at the start of GDB

2.1 File

Specifies the program to be debugged

Syntax

```
file <FILE>
file <path/to/the/file>
```

Description

- Use FILE as a program to be debugged.
- The FILE is read for its symbols, for getting the contents of the memory.
- If FILE cannot be found as specified, your execution directory path ($PATH) is searched for a command of that name.

Known Restrictions

N/A

Example

In this example the file hello.shakti is read from the `/scratch2/shakti/shakti-sdk/software/Example/uart_applns/hello/output/` directory

```
(gdb) file
/scratch2/shakti/shakti-sdk/software/Example/uart_applns/hello/output/hello.shakti
Reading symbols from
/scratch2/shakti/shakti-sdk/software/Example/uart_applns/hello/output/hello.shakti...
(gdb)
```

2.2 Load

Loads the program into memory of the target system and prepares for execution. Used after the file command.

Syntax

```
load
```

Description
N/A

Known Restrictions
N/A

Example

```
(gdb) load
Loading section .text.init, size 0x1c2 lma 0x80000000
Loading section .tohost, size 0x48 lma 0x80001000
Loading section .text, size 0x1018 lma 0x80001048
Loading section .rodata, size 0x1b0 lma 0x80002060
Start address 0x80000000, load size 5074
Transfer rate: 11 KB/sec, 1268 bytes/write.
```

2.3 Target Remote

A remote system connected to gdb via a serial line or network connection. This command tells gdb to use its own remote protocol over a medium for debugging. In the current case, we are debugging the program in the local machine.

Syntax

```
target remote [address:port number]
```

Description

- `address`, address of the machine in which the program is running
- `port, number` used to debug

Known Restrictions
N/A

Example

```
(gdb) target remote localhost:3333
```

3. Commands for executing programs using GDB

3.1 Continue command

Continues running the program until the next breakpoint or error.

Syntax

```
continue / c
```

Description

NA

Known Restrictions

NA

Example

```
(gdb) c
Continuing.
```

3.2 Jump command

Jumps to the specified address. The debugger resumes program execution at that point unless it encounters a breakpoint there.

Syntax

```
jump <*addr>
```

Description

`addr`	Address of the instruction to jump to.

Known Restrictions

NA

Example

```
(gdb) jump *0x1001002e
Continuing at 0x1001002e.
```

3.3 Step command

Steps from one function to another

Syntax

```
step / s
```

Description

NA

Known Restrictions

NA

Example

```
(gdb) load
Loading section .text, size 0x44 lma 0x10010000
Loading section .data, size 0x10 lma 0x10010044
Start address 0x0000000010010000, load size 84
Transfer rate: 2 KB/sec, 42 bytes/write.
(gdb) s
Single stepping until exit from function _start,
which has no line number information.
0x000000001001002e in loop ()
(gdb) s
Single stepping until exit from function loop,
which has no line number information.
0x0000000010010000 in _start ()
(gdb) s
Single stepping until exit from function _start,
which has no line number information.
0x000000001001002e in loop ()
(gdb)
```

3.4 Stepi command

Steps into assembler instructions, **into any function calls.**

Syntax

```
stepi / si
```

Description

- This command executes a machine instruction. When the instruction contains a function call, the command steps into the function being called.
- For multithreaded applications, the stepi command is used to step inside the current thread one machine instruction, while putting all other threads on hold.

Known Restrictions

NA

Example

```
(gdb) file software/Example/uart_applns/hello/output/hello.shakti
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Reading symbols from
software/Example/uart_applns/hello/output/hello.shakti...
(gdb) load
Loading section .text.init, size 0x5e lma 0x80000000
Loading section .text, size 0x3eaa lma 0x80000060
Loading section .rodata, size 0xeb8 lma 0x80003f10
Loading section .eh_frame, size 0x3c lma 0x80004dc8
Loading section .sdata, size 0x28 lma 0x80004e04
Start address 0x80000000, load size 20004
Transfer rate: 5 KB/sec, 3334 bytes/write.
(gdb) stepi
0x80000002 in _start ()
(gdb) si
0x80000004 in _start ()
(gdb) si
0x80000006 in _start ()
```

3.5 Set commands

Set commands changes the value associated with a variable, memory address, or expression that is accessible according to the scope and visibility rules

3.5.1 Set variable command

Sets the value to the variable

Syntax

```
set variable = [expression/value]
```

Description

The **`set variable`** command evaluates the specified expression and assigns the new value.

Known Restrictions

- If you do not specify expression/value, debugger variables are set to void, while program variables do not change.

Example

We load a program sample.elf in the spiking folder.

```
(gdb) file sample.elf
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Load new symbol table from "sample.elf"? (y or n) y
Reading symbols from sample.elf...
(gdb) load
Loading section .text, size 0xae lma 0x10010000
Loading section .data, size 0xa4 lma 0x100100b0
Loading section .sdata, size 0x4 lma 0x10010154
Start address 0x0000000010010000, load size 342
Transfer rate: 3 KB/sec, 114 bytes/write.
(gdb) si
0x0000000010010004    9    while (flag);
(gdb) si
0x0000000010010008    9    while (flag);
(gdb) si
0x000000001001000a    9    while (flag);
(gdb) print string
$1 = "This Is $hakti, The 1st Ever 'Made In India' Computer Chip"
(gdb) print flag
$2 = 1
(gdb) set flag =2
(gdb) print flag
$3 = 2
(gdb)
```

3.5.2 Set value in a memory mapped register

Sets the assigned value to memory location or memory mapped register

Syntax

```
set <address> = <value>
```

Description

NA

Known Restrictions

NA

Example

```
(gdb) file ./software/examples/uart_applns/hello/output/hello.shakti
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Reading symbols from ./software/examples/uart_applns/hello/output/hello.shakti...
(gdb) load
Loading section .text.init, size 0x5e lma 0x80000000
```

```
Loading section .text, size 0x3730 lma 0x80000060
Loading section .rodata, size 0xfc8 lma 0x80003790
Loading section .sdata, size 0x58 lma 0x80004758
Start address 0x0000000080000000, load size 18350
Transfer rate: 6 KB/sec, 4587 bytes/write.
(gdb) set *((char *)0x11304)=67 #set the address to char value 67 (alphabet C)
(gdb) x/x 0x11304 #Printing the hex equivalent in the address
0x11304:   0x00000043
(gdb) x/c 0x11304
0x11304:   67 'C'#Printing the char value in the address
(gdb)
```

---

4. Commands for Break points

4.1 Break command

Creates a breakpoint at a specified line, address

Syntax

```
b
break [Function Name]
break [Line Number]
break [Address]
```

Description

- **`Function Name`**, when specified, the `break` command will set a breakpoint at the beginning of the specified function
- **`Line Number`**, when specified, the `break` command will set a breakpoint at a given location inside the specified file. If no file is specified, the current source file will be used.
- **`Address`**, when specified, the `break` command will set a breakpoint at a given address. The address should be a valid C/C++ expression (e.g. a hexadecimal number starting with 0x).

Known Restrictions

Do not confuse the **`[Function name]`** and the **`[Address]`** forms of the break command. For instance, If you attempt to set a breakpoint at *`address`* **`0x40138c`**, the following command will fail,

```
(gdb)break 0x80001074
```

This happens because GDB will interpret *0x80001074* as a function name rather than an address. The correct syntax to set a breakpoint at address **0x80001074** is,

```
(gdb)break *0x8000107
```

Example

```
(gdb) b start
Breakpoint 1 at 0x80000c78: file
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c, line 85.
(gdb) b main
Breakpoint 2 at 0x80000068: file ./ds3231_sensor.c, line 61.
(gdb) b *0x80000f6e
Breakpoint 3 at 0x80000f6e: file
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c, line 180.
(gdb) b *0x80001074
Breakpoint 4 at 0x80001074: file
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c, line 224.
(gdb) b *0xfef407a3
Breakpoint 5 at 0xfef407a3
(gdb)
#Five breakpoints set at start, main and addresses 0x80000f6e,0x80001074 &
0xfef407a3
```

4.2 Info breakpoints command

Displays information about breakpoints.

Syntax

```
info breakpoints
info breakpoints [Numbers]
```

Description

*Number, w*hen the **info breakpoints** command is invoked without any arguments, it displays information about all breakpoints. When one or more numbers are specified, only the breakpoints matching the specified numbers will be displayed.

Known Restrictions

NA

Example

```
(gdb) info breakpoints
No breakpoints or watchpoints.

(gdb) b main
Breakpoint 1 at 0x80000068: file ./ds3231_sensor.c, line 61.
(gdb) b start
Breakpoint 2 at 0x80000c78: file
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c, line 85.
(gdb) b *0x800033b2
Breakpoint 3 at 0x800033b2: file
/home/shakti-sdk-dev/bsp/drivers/plic/plic_driver.c, line 441.

(gdb) info breakpoint
Num   Type            Disp Enb Address           What
1     breakpoint      keep y   0x0000000080000068 in main at
./ds3231_sensor.c:61
2     breakpoint      keep y   0x0000000080000c78 in start
                                                  at
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c:85
3     breakpoint      keep y   0x00000000800033b2 in configure_interrupt
                                                  at
/home/shakti-sdk-dev/bsp/drivers/plic/plic_driver.c:441
(gdb)
```

4.3 Clear command

Deletes a breakpoint at a specified location.

Syntax

```
clear [Location]
```

Description

Specifies the `location` of a breakpoint that should be removed. If not specified, GDB will try to remove a breakpoint in the current location.

Known Restrictions

NA

Example

```
(gdb) info breakpoint #displays the breakpoint available
Num   Type            Disp Enb Address           What
11    breakpoint      keep y   0x0000000080000c78 in start
```

```
                                                    at
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c:85
12    breakpoint      keep y   0x00000000fef407a3

(gdb) clear *0x00000000fef407a3 #deletes breakpoint 12

(gdb) info breakpoint
Deleted breakpoint 12 Num   Type            Disp Enb Address           What
11    breakpoint      keep y   0x0000000080000c78 in start at
      /home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c:85
(gdb)
```

4.4 Disable command

Disables all breakpoints or specified breakpoints.

Syntax

```
disable
disable [Number]
disable [Number1] [Number2] ... [NumberN]
disable breakpoints ...
```

Description

When the `disable` command is invoked without any arguments, it disables all the breakpoints. When one or more numbers are specified, only the breakpoints matching the specified numbers will be disabled.

Known Restrictions

NA

Example

```
(gdb) info breakpoint
No breakpoints or watchpoints.
(gdb) b I2cStop
Breakpoint 1 at 0x80001556: file
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c, line 440.
(gdb) b main
Breakpoint 2 at 0x80000068: file ./ds3231_sensor.c, line 61.
(gdb) b *0x80001b0c
Breakpoint 3 at 0x80001b0c: file /home/shakti-sdk-dev/bsp/libs/util.c, line
209.
(gdb) b *0x800034c4
Breakpoint 4 at 0x800034c4: file
/home/shakti-sdk-dev/bsp/drivers/pwm/pwm_driver.c, line 93.
```

```
(gdb) info breakpoint
Num   Type            Disp Enb Address           What
1     breakpoint      keep y   0x0000000080001556 in I2cStop
                                                  at
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c:440
2     breakpoint      keep y   0x0000000080000068 in main at
./ds3231_sensor.c:61
3     breakpoint      keep y   0x0000000080001b0c in delay_loop
                                                  at
/home/shakti-sdk-dev/bsp/libs/util.c:209
4     breakpoint      keep y   0x00000000800034c4 in set_pwm_control_register
                                                  at
/home/shakti-sdk-dev/bsp/drivers/pwm/pwm_driver.c:93

(gdb) disable 2 #disables one breakpoint

(gdb) info breakpoint
Num   Type            Disp Enb Address           What
1     breakpoint      keep y   0x0000000080001556 in I2cStop
                                                  at
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c:440
2     breakpoint      keep n   0x0000000080000068 in main at
./ds3231_sensor.c:61
3     breakpoint      keep y   0x0000000080001b0c in delay_loop
                                                  at
/home/shakti-sdk-dev/bsp/libs/util.c:209
4     breakpoint      keep y   0x00000000800034c4 in set_pwm_control_register
                                                  at
/home/shakti-sdk-dev/bsp/drivers/pwm/pwm_driver.c:93

(gdb) disable #disables all the breakpoints

(gdb) info breakpoint
Num   Type            Disp Enb Address           What
1     breakpoint      keep n   0x0000000080001556 in I2cStop
                                                  at
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c:440
2     breakpoint      keep n   0x0000000080000068 in main at
./ds3231_sensor.c:61
3     breakpoint      keep n   0x0000000080001b0c in delay_loop
                                                  at
/home/shakti-sdk-dev/bsp/libs/util.c:209
4     breakpoint      keep n   0x00000000800034c4 in set_pwm_control_register
                                                  at
/home/shakti-sdk-dev/bsp/drivers/pwm/pwm_driver.c:93
```

```
(gdb)
```

4.5 Delete command

Deletes specific breakpoints or all breakpoints.

Syntax

```
delete
delete [number]
```

Description

- **`number`,** When the **delete** is invoked without any arguments, it deletes all breakpoints. When one or more numbers are specified, only the breakpoints matching the specified numbers will be deleted.

Known Restrictions

NA

Example

```
(gdb) info breakpoints
Num   Type            Disp Enb Address           What
3     breakpoint      keep y   <PENDING>         start
4     breakpoint      keep y   <PENDING>         0x8000022c
5     breakpoint      keep y   0x000000008000009c in main at ./uart_test.c:51
6     breakpoint      keep y   <PENDING>         0x8000045c
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) info breakpoints
No breakpoints or watchpoints.
```

4.6 Enable command

Enables all breakpoints or specified breakpoints.

Syntax

```
enable
enable [Number]
enable [Number1] [Number2] ... [NumberN]
```

Description

- **Number**, when the **enable** is invoked without any arguments, it enables all breakpoints. When one or more numbers are specified, only the breakpoints matching the specified numbers will be enabled.

Known Restrictions

NA

Example

```
(gdb) info breakpoint
Num   Type            Disp Enb Address           What
1     breakpoint      keep n   0x0000000080001556 in I2cStop
                                                  at
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c:440
2     breakpoint      keep n   0x0000000080000068 in main at
./ds3231_sensor.c:61
3     breakpoint      keep n   0x0000000080001b0c in delay_loop
                                                  at
/home/shakti-sdk-dev/bsp/libs/util.c:209
4     breakpoint      keep n   0x00000000800034c4 in set_pwm_control_register
                                                  at
/home/nambirajan/shakti-sdk-dev/bsp/drivers/pwm/pwm_driver.c:93

(gdb) enable 4 #enables breakpoint no:4

(gdb) info breakpoint
Num   Type            Disp Enb Address           What
1     breakpoint      keep n   0x0000000080001556 in I2cStop
                                                  at
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c:440
2     breakpoint      keep n   0x0000000080000068 in main at
./ds3231_sensor.c:61
3     breakpoint      keep n   0x0000000080001b0c in delay_loop
                                                  at
/home/shakti-sdk-dev/bsp/libs/util.c:209
4     breakpoint      keep y   0x00000000800034c4 in set_pwm_control_register
                                                  at
/home/nambirajan/shakti-sdk-dev/bsp/drivers/pwm/pwm_driver.c:93

(gdb) enable #enables all the breakpoints

(gdb) info breakpoint
Num   Type            Disp Enb Address           What
1     breakpoint      keep y   0x0000000080001556 in I2cStop
                                                  at
/home/shakti-sdk-dev/bsp/drivers/i2c/gpio_i2c.c:440
```

```
2     breakpoint      keep y   0x0000000080000068 in main at
./ds3231_sensor.c:61
3     breakpoint      keep y   0x0000000080001b0c in delay_loop
                                                  at
/home/shakti-sdk-dev/bsp/libs/util.c:209
4     breakpoint      keep y   0x00000000800034c4 in set_pwm_control_register
                                                  at
/home/shakti-sdk-dev/bsp/drivers/pwm/pwm_driver.c:93
(gdb)
```

---

5. Commands to Display information

5.1 Display command

Displays the memory contents at a given address using the specified format.

Syntax

```
x /[Format] [Expression]
x [Address expression]
x /[Length][Format] [Address expression]
```

Description

`Expression`, specifies the expression that will be automatically evaluated and displayed after each step.

Known Restrictions

NA

Format

If specified, allows overriding the output format used by the command. Valid format specifiers are:

- o - octal
- x - hexadecimal
- u - unsigned decimal
- t - binary
- f - floating point
- a - address
- c - char
- s - string

Example

```
(gdb) x/wx 0x80000000
0x80000000 <main>:    0xec221101
(gdb) x 0x80000000
0x80000000 <main>:    0xec221101
(gdb) x/c 0x80000000
0x80000000 <main>:    1 '\001'
(gdb) x/wx 0x80000000
0x80000000 <main>:    0xec221101
(gdb) x/2hx 0x80000000
0x80000000 <main>:    0x1101    0xec22
(gdb) x/gx 0x80000000
0x80000000 <main>:    0x87a21000ec221101
(gdb) x/s 0x80000000
0x80000000 <main>:    "\001\021", <incomplete sequence \354>
(gdb) x/5bx 0x80000000
0x80000000 <main>:    0x01    0x11    0x22    0xec    0x00
(gdb) x/10x 0x8000084a
0x8000084a <_printf_+832>:    0x70e60001    0x74a67446    0x61097906
0x71598082
0x8000085a <printf+2>:    0xf022f406    0x3c231800    0xe40cfca4
0xec14e810
0x8000086a <printf+18>:    0xf41cf018    0x03043823
(gdb)
```

5.2 Print command

Prints the value of a given expression.

Syntax

```
Print [Expression]
Print {[type]}[Address]
```

Description

- **`Expression,`** specifies the expression that will be evaluated and printed. The expression can include references to variables (e.g. i), registers (e.g. $eax) and pseudo-registers (e.g. $pc).
- Note that if your **`expression`** refers to a local variable, you need to ensure that you have selected the correct frame.
- **`Type/Address,`** this format allows explicitly specifying the address of the evaluated expression and can be used as a shortcut to the C/C++ type conversion. E.g. ***((int *)p)** is equivalent to **{int}p**

Known Restrictions

NA

Example

```
(gdb) print loop
$1 = {<text variable, no debug info>} 0x1001002e <loop>
(gdb)
$2 = {<text variable, no debug info>} 0x1001002e <loop>
(gdb) print *0x1001001e
$3 = 1943
(gdb)
```

5.3 Info address command

Displays the address of a given symbol

Syntax

```
info address [Symbol name]
```

Description

- **`Symbol name.`** specifies the symbol (function or variable) whose address should be displayed.
- The **`info address`** command produces similar output to the **`print&`** command. However, unlike the `print` command it does not display the type information, but prints whether the symbol is a function or a variable.
- Note that the program does not need to be running in order to use the info address command.

- In order to do the reverse operation (get a symbol name from address), use the **`info symbol`** command.

Known Restrictions
NA

Example
```
(gdb) info address main
Symbol "main" is a function at address 0x80000098.
(gdb)  print &main
$1 = (int (*)(void)) 0x80000098 <main>
(gdb)
```

5.4 Info registers command

Displays the contents of all processor registers

Syntax
```
info reg / i r / info registers
info all-registers / i all-registers
info registers [Register name]
```

Description
- **`info reg / i r / info registers`**, displays all the register values **except the CSR registers and the floating point** registers.
- **`info all-registers`**, displays all the register values **including the CSR registers and the floating point** registers.
- **`info registers [Register name]`**, displays the value of the **specified register**.

Known Restrictions
NA

Example
```
(gdb) i r
ra             0x0    0x0
sp             0x0    0x0
gp             0x0    0x0
tp             0x0    0x0
t0             0x10010000    268500992
t1             0x0    0
t2             0x0    0
fp             0x0    0x0
s1             0x0    0
a0             0x0    0
```

```
a1             0x1020    4128
a2             0x0    0
a3             0x0    0
a4             0x1    1
a5             0x10010000    268500992
a6             0x0    0
a7             0x0    0
s2             0x0    0
s3             0x0    0
s4             0x0    0
s5             0x0    0
s6             0x0    0
s7             0x0    0
s8             0x0    0
s9             0x0    0
s10            0x0    0
s11            0x0    0
t3             0x0    0
t4             0x0    0
t5             0x0    0
t6             0x0    0
pc             0x10010008    0x10010008 <main+8>
(gdb) i r a0
a0             0x0000000000000000    0
(gdb) i r mie
mie            0x60000    393216
(gdb)
```

5.5 Info functions command

Displays the list of functions in the debugged program

Syntax

```
info functions
info functions [Regex]
```

Description

- `Regex, if` specified, the `info functions` command will list the functions matching the regular expression `regex`.. If omitted, the command will list all functions in all loaded modules (main program and shared libraries)
- Note that running the **info functions** command without arguments can produce a lot of output as the list of all functions in all loaded shared libraries is typically very long.

Known Restrictions

NA

Example

```
(gdb) info functions
All defined functions:

File ./uart_test.c:
int main(void);

File
/home/risecreek/users/gitlab/shakti-sdk/shakti-tools/riscv64-unknown-elf/incl
ude/stdio.h:
int printf(const char *, ...);

File /home/risecreek/users/username/drivers/05_06_2020/sw/bsp/core/init.c:
void init(void);
void section_init();
void trap_init();

File /home/risecreek/users/drivers/05_06_2020/sw/bsp/core/traps.c:
void default_handler(uintptr_t, uintptr_t);
uintptr_t handle_trap(uintptr_t, uintptr_t);
---Type <return> to continue, or q <return> to quit---


(gdb) info functions printf
All functions matching regular expression "printf":

File
/home/risecreek/users/gitlab/shakti-sdk/shakti-tools/riscv64-unknown-elf/incl
ude/stdio.h:
int printf(const char *, ...);

File /home/risecreek/users/drivers/05_06_2020/iisu/sw/bsp/libs/printf.c:
void _printf_(const char *, va_list);
(gdb)
```

5.6 Info line command

Translates line numbers to addresses and vice versa.

Syntax

```
info line [File]:[Line]
info line [Function]
info line [File]:[Function]
info line *[Address]
```

Description

- `File:Line,` if this form is used, the **info line** command will display the starting and ending addresses of the given source line.
- `Function,` if this parameter is used, the **info line** command will display the information about the first source line of a given function.
- `Address,` if this parameter is used, the **info line** command will display the information about a source line that corresponds to the specified address.

Known Restrictions

NA

Example

```
(gdb) info line *0x80000a44
Line 227 of "/home/username/shakti-sdk/bsp/drivers/uart/uart.c" starts at
address 0x80000a3e <set_uart_rx_threshold> and ends at 0x80000a4e
<set_uart_rx_threshold+16>.
(gdb)
(gdb)
(gdb) info line main
Line 49 of "./uart_test.c" starts at address 0x80000098 <main> and ends at
0x800000a0 <main+8>.
(gdb)
```

5.7 Info variables command

Displays the list of global/static variables present in the debugged program

Syntax

```
info variables
info variables [Regex]
```

Parameters

`Regex, if` specified, the **info variables** command will list the global/static variables matching the regex. If omitted, the command will list all global/static variables in all loaded modules (main program and shared libraries).

Example

```
(gdb) info variables
All defined variables:
File /home/username/shakti-sdk/bsp/core/init.c:
char *bss_end;
char *bss_start;
char *heap_end;
char *heap_start;
char *sbss_end;
char *sbss_start;
char *stack_end;
char *stack_start;

File /home/username/shakti-sdk/bsp/drivers/clint/clint_driver.c:
uint64_t *mtime;
uint64_t *mtimecmp;
---Type <return> to continue, or q <return> to quit---

(gdb) info variables start
All variables matching regular expression "start":

File /home/username/shakti-sdk/bsp/core/init.c:
char *bss_start;
char *heap_start;
char *sbss_start;
char *stack_start;

Non-debugging symbols:
0x0000000080002e38  __rodata_start
0x0000000080003c98  __data_start
```

5.8 Info symbol command

Displays the name of the symbol residing at a given address

Syntax

```
info symbol [Address]
info symbol [Expression]
```

Description

`Address`, specifies the address that will be searched for a symbol.
`Expression`, specifies an expression that will be evaluated to get the address. The expression can contain pseudo-registers

Known Restrictions

NA

Example

```
(gdb) info symbol main
main in section .text
(gdb) print main
$1 = {<text variable, no debug info>} 0x80000000 <main>
(gdb) info symbol 0x80000000
main in section .text
```

5.9 Info types command

Displays the list of types defined in the currently loaded modules

Syntax

```
info types
info types [Regex]
```

Description

`Regex,` if specified, the **info types** command will list the types matching the regex. If omitted, the command will list all types in all loaded modules (main program and shared libraries).

Known Restrictions

NA

Example

```
(gdb) info types
All defined types:

Non-debugging symbols:
0x0000000003fffdf6  _HEAP_SIZE
0x0000000003fffdf6  _STACK_SIZE
(gdb) info types [Regex]
All types matching regular expression "[Regex]":

File ../../../../riscv-gcc/libgcc/../gcc/config/riscv/riscv-opts.h:
24:    enum riscv_abi_type;
```

```
35:    enum riscv_code_model;
43:    enum riscv_microarchitecture_type;

File ../../../../riscv-gcc/libgcc/../include/hashtab.h:
54:    typedef int (*)(const void *, const void *) htab_eq;

File ../../../../riscv-gcc/libgcc/libgcc2.c:
    complex double
    complex float
    complex long double
    double
    long double
    long
    long long
    unsigned long long
    unsigned long
    unsigned short
    signed char
    unsigned char
--Type <RET> for more, q to quit, c to continue without paging--
    unsigned int

File ../../../../riscv-gcc/libgcc/libgcc2.h:
132:    typedef long long DItype;
128:    typedef int SItype;
133:    typedef unsigned long long UDItype;
123:    typedef unsigned char UQItype;
129:    typedef unsigned int USItype;
```