GNU Debugger (gdb) in the arm cross-toolchain

1. **include the debugging information section**

```
MINGW32:/c/Users/Hp/Desktop/Lab 1- Lesson 2

learn-in-depth.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .startup      00000010  00010000  00010000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text         00000068  00010010  00010010  00008010  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .data         00000034  00010078  00010078  00008078  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .ARM.attributes 0000002e  00000000  00000000  000080ac  2**0
                  CONTENTS, READONLY
  4 .comment      00000011  00000000  00000000  000080da  2**0
                  CONTENTS, READONLY
  5 .debug_line   000000ac  00000000  00000000  000080eb  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_info   0000011e  00000000  00000000  00008197  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_abbrev 000000bf  00000000  00000000  000082b5  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_aranges 00000060  00000000  00000000  00008378  2**3
                  CONTENTS, READONLY, DEBUGGING
  9 .debug_loc    00000058  00000000  00000000  000083d8  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .debug_str    0000006a  00000000  00000000  00008430  2**0
                  CONTENTS, READONLY, DEBUGGING
 11 .debug_frame  00000054  00000000  00000000  0000849c  2**2
                  CONTENTS, READONLY, DEBUGGING

Hp@RawanSleem MINGW32 ~/Desktop/Lab 1- Lesson 2
$
```

1. **Set up the virtual board for a debugging session**

```
MINGW32:/c/Users/Hp/Desktop/Lab 1- Lesson 2

p@RawanSleem MINGW32 ~/Desktop/Lab 1- Lesson 2
 D:/qemu/qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-
n-depth.elf
```

2. **Set up the connection between the gdb in the local host and the gdb server in the virtual machine.**
   The virtual machine is connected to the ethernet card on the local host machine, so the remote target will be the local host IP address, with a port number of 1234 by default.

```
MINGW32:/c/Users/Hp/Desktop/Lab 1- Lesson 2

Hp@RawanSleem MINGW32 ~/Desktop/Lab 1- Lesson 2
$ export PATH="/d/ARM_TOOLCHAIN/bin/:$PATH"

Hp@RawanSleem MINGW32 ~/Desktop/Lab 1- Lesson 2
$ arm-none-eabi-gdb.exe learn-in-depth.elf
GNU gdb (GDB) 7.5.1
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-mingw32 --target=arm-none-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from C:\Users\Hp\Desktop\Lab 1- Lesson 2\learn-in-depth.elf...do
ne.
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
reset () at startup.s:4
4       ldr sp, = stack_top
(gdb)
```

The debugger points at the first address specified for the startup in the linker script

3. **To display a specified number of instructions and the PC location**

```
(gdb) display/3i $pc
1: x/3i $pc
=> 0x10000 <reset>:     ldr     sp, [pc, #4]    ; 0x1000c <stop+4>
   0x10004 <reset+4>:   bl      0x10060 <main>
   0x10008 <stop>:      b       0x10008 <stop>
(gdb)
```

4. **Put breakpoints at specified locations or functions**

```
(gdb) b main
Note: breakpoint 1 also set at pc 0x10068.
Breakpoint 3 at 0x10068: file app.c, line 6.
(gdb) b *0x10060
Breakpoint 4 at 0x10060: file app.c, line 4.
(gdb) b uart_tx
Breakpoint 5 at 0x10020: file uart.c, line 6.
(gdb)
```

5. **To delete and show the current breakpoints**

```
5       breakpoint     keep y   0x00010020 in uart_tx at uart.c:6
(gdb) delet breakpoint 1
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
3       breakpoint     keep y   0x00010068 in main at app.c:6
4       breakpoint     keep y   0x00010060 in main at app.c:4
5       breakpoint     keep y   0x00010020 in uart_tx at uart.c:6
(gdb) |
```

6. **To continue**

```
(gdb) c
Continuing.

Breakpoint 4, main () at app.c:4
4       void main(void){
1: x/3i $pc
=> 0x10060 <main>:      push    {r11, lr}
   0x10064 <main+4>:    add     r11, sp, #4
   0x10068 <main+8>:    ldr     r0, [pc, #4]    ; 0x10074 <main+20>
(gdb) c
Continuing.

Breakpoint 5, uart_tx (p_string=0x10078 <string> "Learn-In-Depth: Rawan")
    at uart.c:6
6               while(*p_string != 0){
1: x/3i $pc
=> 0x10020 <uart_tx+16>:        b       0x10040 <uart_tx+48>
   0x10024 <uart_tx+20>:        ldr     r3, [pc, #48]   ; 0x1005c <uart_tx+76>
   0x10028 <uart_tx+24>:        ldr     r2, [r11, #-8]
(gdb) c
Continuing.
```

7. **To jump one assembly line or one C line (which can be multiple assembly lines)**

```
(gdb) si
reset () at startup.s:5
5       bl main
(gdb) si
main () at app.c:4
4       void main(void){
(gdb) s
6               uart_tx(string);
(gdb) |
```

8. **To watch a variable and if a certain change in its value caused an error**

```
(gdb) watch string[0]
Hardware watchpoint 1: string[0]
(gdb) |
```