CO224 : Computer Architecture

Using GDB for ARM

Install *gcc- arm-linux-gnueabihf* from the following link.

https://launchpad.net/linaro-toolchain-binaries/trunk/2012.10

For Linux you will have to select the package named *gcc-linaro-arm-linux-gnueabihf-4.7-2012.10-20121022_linux.tar.bz2*

Previously we used *gcc- arm-linux-gnueabi* while *gnueabihf* is a newer tool chain that supports some floating point features in new arm processors. You can install *gcc- arm-linux-gnueabihf* from *apt-get install* as well, but the problem is it will not have *gdb*.

Installation steps are as follows:

- Download the file *gcc-linaro-arm-linux-gnueabihf-4.7-2012.10-20121022_linux.tar.bz2* from the link.
- Extract the tar file

  tar -xvf gcc-linaro-arm-linux-gnueabihf-4.7-2012.10-20121022_linux.tar.bz2

- Move the extracted folder to some location.

  sudo mv gcc-linaro-arm-linux-gnueabihf-4.7-2012.10-20121022_linux /usr/local/arm-linux-gnueabihf

- The binaries are located in /usr/local/ arm-linux-gnueabihf/bin. We have to add this to the path variable.

  PATH=$PATH: /usr/local/arm-linux-gnueabihf/bin

  Path added in this method is not be persistent across reboots. If you want so, consider adding the path to */etc/environment*

If you are lazy to install you can simply use the tesla server as the things are already installed there.

Debugging steps are as follows:

- If everything is fine you will be able to compile using the following commands.

  arm-linux-gnueabihf-gcc source.s -o binary -g

  Here –g is for the debugger

- Run the program through *qemu* as follows

  qemu-arm -g 12345 -L /usr/local/arm-linux-gnueabihf/arm-linux-gnueabihf/libc/ binary

  -g is to tell qemu-arm to open a port on 12345 and wait for the debugger

- Now take another terminal and launch *gdb* as follows

  arm-linux-gnueabihf-gdb binary

- Inside *gdb*, set the port as follows

  Target remote localhost:12345

  The debugger connects to the *qemu* instance. Any input to be given or the output to be viewed is to be done through *qemu* while *gdb* is just to issue debug commands. This is a bit different to the usual *gdb* we know, where input and output is given via the *gdb* itself.

- Now enter a breakpoint at the main function as follows.

  break main

- Now give the command *continue.*

- Then as usual you can use commands such as break, step, next and continue.

Some useful gdb commands

- The ***step*** command does a single step. While something **like *step 10*** executes 10 more steps. ***next*** command can be used instead of step ***when*** required to step over function calls.

```
(gdb) break main
Breakpoint 1 at 0x8404: file strlen.s, line 17.
(gdb) continue
Continuing.

Breakpoint 1, main () at strlen.s:17
17              str     lr, [sp, #100]
(gdb) step
20              ldr     r0, =formatr
(gdb) step
21              bl      printf
(gdb) step 10
54              cmp     r2, #0
(gdb) step 10
59              b       loop
(gdb) step
53              ldrb    r2, [r0, #0]
(gdb) continue
Continuing.
[Inferior 1 (Remote target) exited normally]
(gdb) quit
```

- ***info registers*** command will show the values of all registers at the current point.

```
r0             0x1      1
r1             0xf6fff934       -150996684
r2             0xf6fff93c       -150996676
r3             0x8400   33792
r4             0x0      0
r5             0xf6fff7f8       -150997000
r6             0x8329   33577
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0xf67fe000       -159391744
r11            0x0      0
r12            0xf6700cc1       -160428863
sp             0xf6fff780       0xf6fff780
lr             0xf66f2fdb       -160485413
pc             0x8404   0x8404 <main+4>
cpsr           0x60000010       1610612752
```

- To get the value of a specific register for example r0 use the command ***info registers r0*.** You can use the short hand ***i r r0*** as well, for your convenience.
- Further you can use following commands as well to print the contents of the r0 register in decimal, hexadecimal and binary respectively.

  print/d $r0

  print/x $r0

  print/t $r0

- If you need to print the value of a register in each step, for example value of r0 and r1, first issue the commands ***display $r0*** and ***display $r1***. Then each time you use ***step*** command, it displays the value of r0 and r1 as follows.

```
(gdb) display $r0
1: $r0 = 1
(gdb) display $r1
2: $r1 = -150996684
(gdb) step
21              bl      printf
2: $r1 = -150996684
1: $r0 = 67076
(gdb) step
24              ldr     r0, =formats
2: $r1 = 1
1: $r0 = 16
(gdb) step
25              mov     r1, sp
2: $r1 = 1
1: $r0 = 67093
(gdb)
```

- To get the instruction at a certain memory address use the ***disassemble*** command.

```
(gdb) i r pc
pc             0x8414   0x8414 <main+20>
(gdb) disassemble 0x8420
Dump of assembler code for function main:
   0x00008400 <+0>:     sub     sp, sp, #104    ; 0x68
   0x00008404 <+4>:     str     lr, [sp, #100]  ; 0x64
   0x00008408 <+8>:     ldr     r0, [pc, #104]  ; 0x8478 <endLoop+16>
   0x0000840c <+12>:    bl      0x82e8
   0x00008410 <+16>:    ldr     r0, [pc, #100]  ; 0x847c <endLoop+20>
=> 0x00008414 <+20>:    mov     r1, sp
   0x00008418 <+24>:    bl      0x8300
   0x0000841c <+28>:    mov     r0, sp
   0x00008420 <+32>:    bl      0x8444 <stringLen>
   0x00008424 <+36>:    mov     r1, sp
   0x00008428 <+40>:    mov     r2, r0
   0x0000842c <+44>:    ldr     r0, [pc, #76]   ; 0x8480 <endLoop+24>
   0x00008430 <+48>:    bl      0x82e8
   0x00008434 <+52>:    mov     r0, #0
   0x00008438 <+56>:    ldr     lr, [sp, #100]  ; 0x64
   0x0000843c <+60>:    add     sp, sp, #104    ; 0x68
   0x00008440 <+64>:    mov     pc, lr
End of assembler dump.
(gdb)
```

- Memory contents can be examined using the ***x*** command. The ***x*** command is optionally followed by a "/", a count field, a format field, a size field and finally a memory address. The count field is a number in decimal. The format field is a single letter with 'd' for decimal, 'x' for hexadecimal, 't' for binary and 'c' for ASCII. The size field is also a single letter with 'b' for byte, 'h' for 16-bit word (half word) and 'w' for a 32-bit word. For example,

x/12cb 0xf6fff780

x/12db 0xf6fff780

x/12xh 0xf6fff780

x/12xw 0xf6fff780

will print out respectively the contents of memory starting at 0xf6fff780 in the following manner: the next 12 bytes as ASCII characters, the next 12 bytes as decimal numbers, the next 12 16-bit words in hex, and the next 12 32-bit words in hex.

Enjoy Debugging! :P